Non Linear Data Structure

Tree

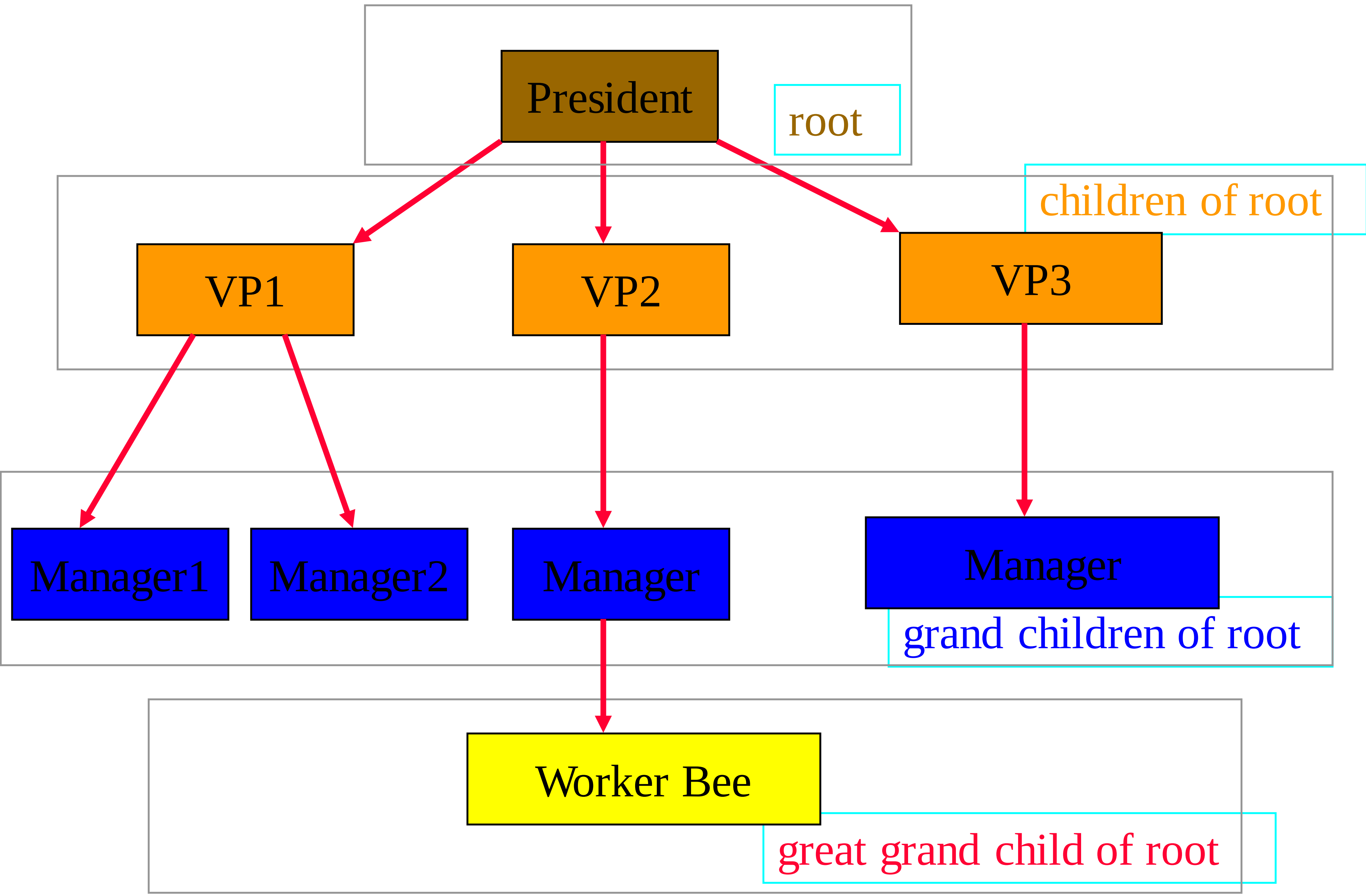
Linear Lists And Trees

- Linear lists are useful for serially ordered data.
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week.
  - Months in a year.
  - Students in this class.
- Trees are useful for hierarchically ordered data.
  - Employees of a corporation.
    - President, vice presidents, managers, and so on.

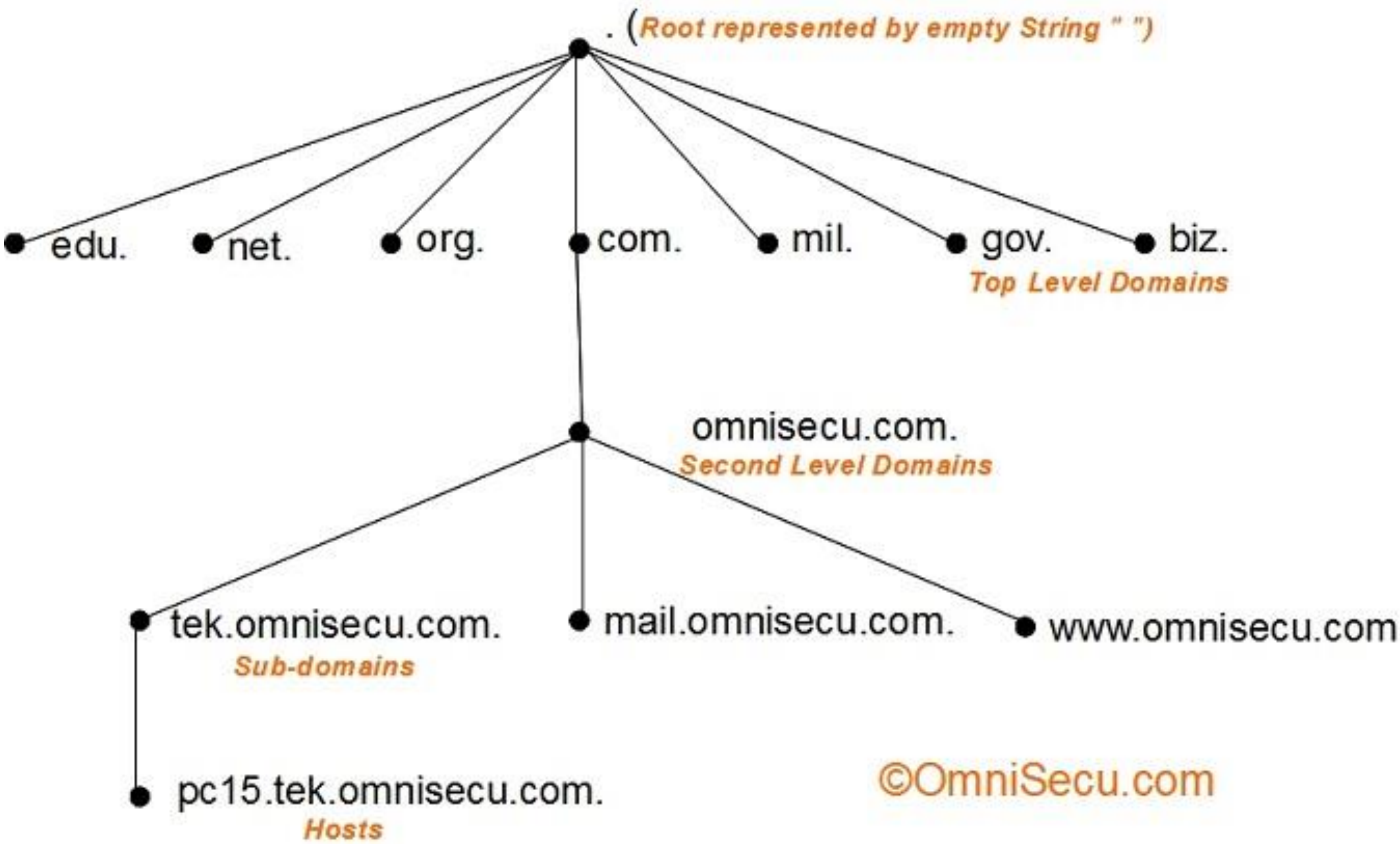
Hierarchical Data And Trees

- The element at the top of the hierarchy is the root.
- Elements next in the hierarchy are the children of the root.
- Elements next in the hierarchy are the grandchildren of the root, and so on.
- Elements that have no children are leaves.

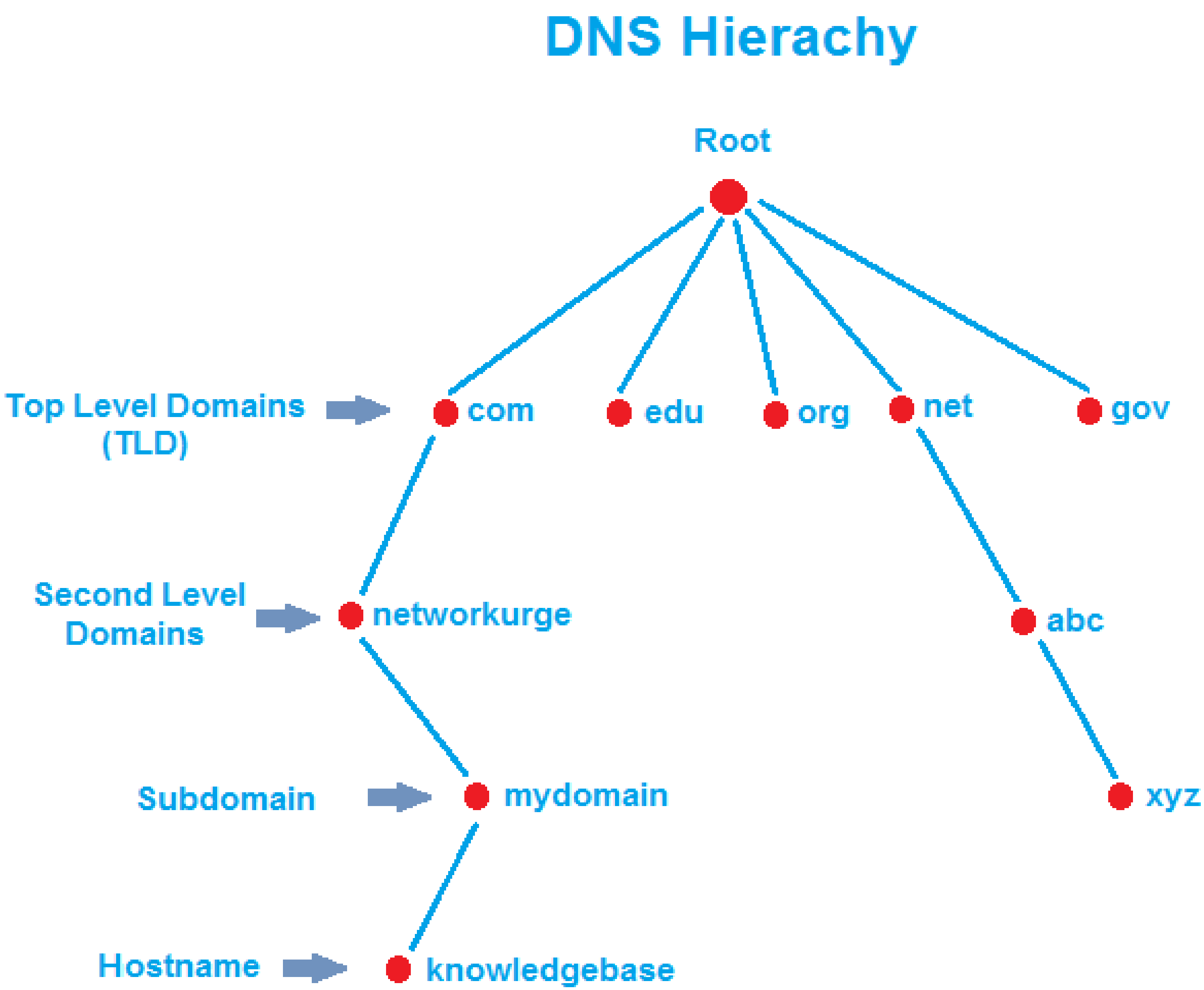
Example Tree

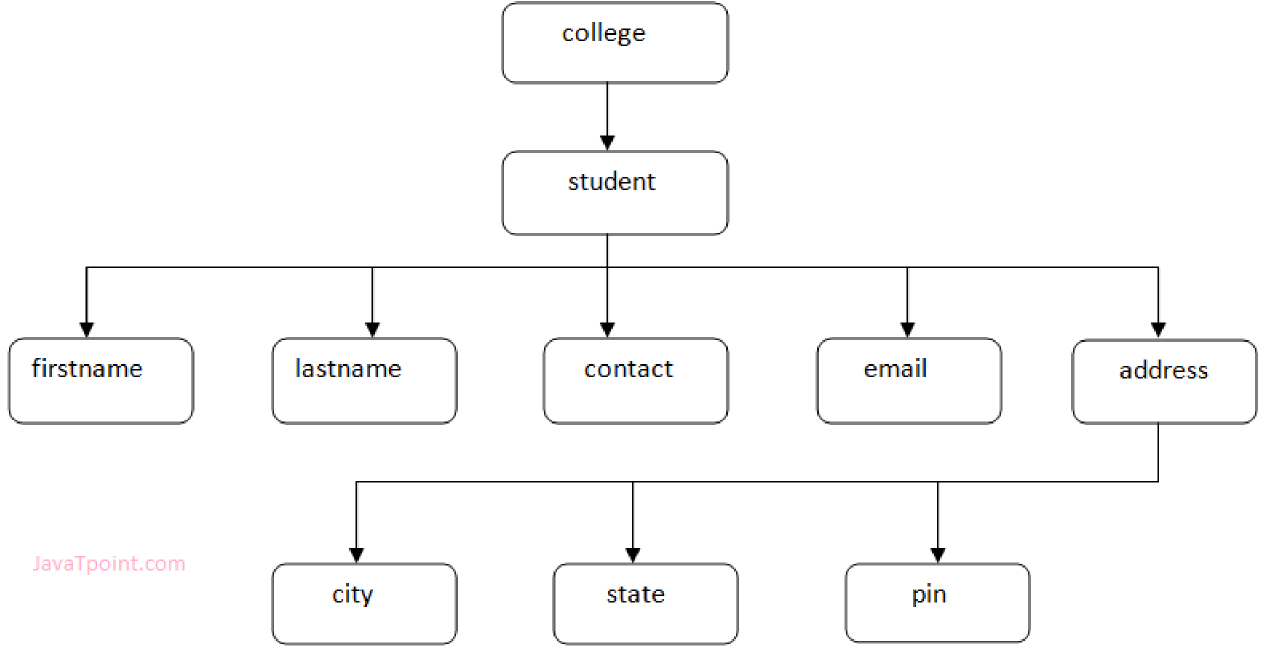


Example Tree- DNS



Example Tree- DNS

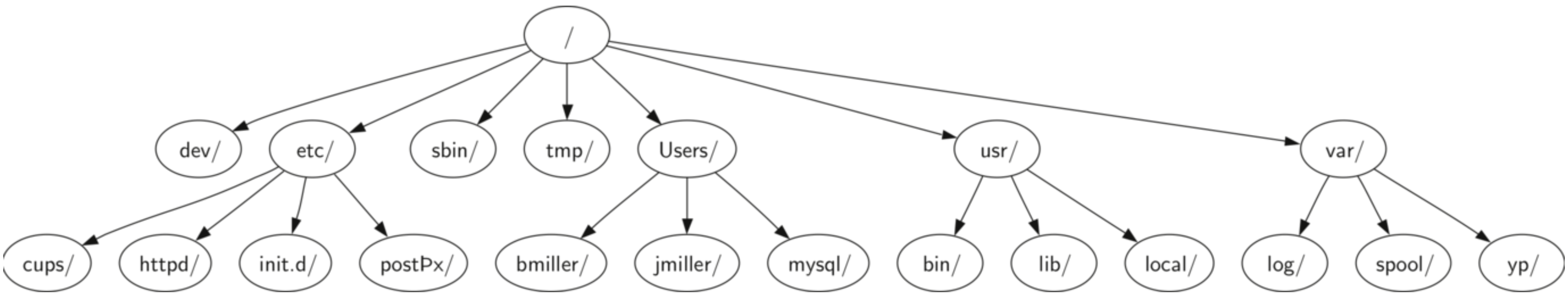




Example Tree-XML Parsing

```
<?xml version="1.0"?>
<college>
 <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
        <city>Ghaziabad</city>
        <state>Uttar Pradesh</state>
        <pin>201007</pin>
    </address>
 </student>
</college>
```

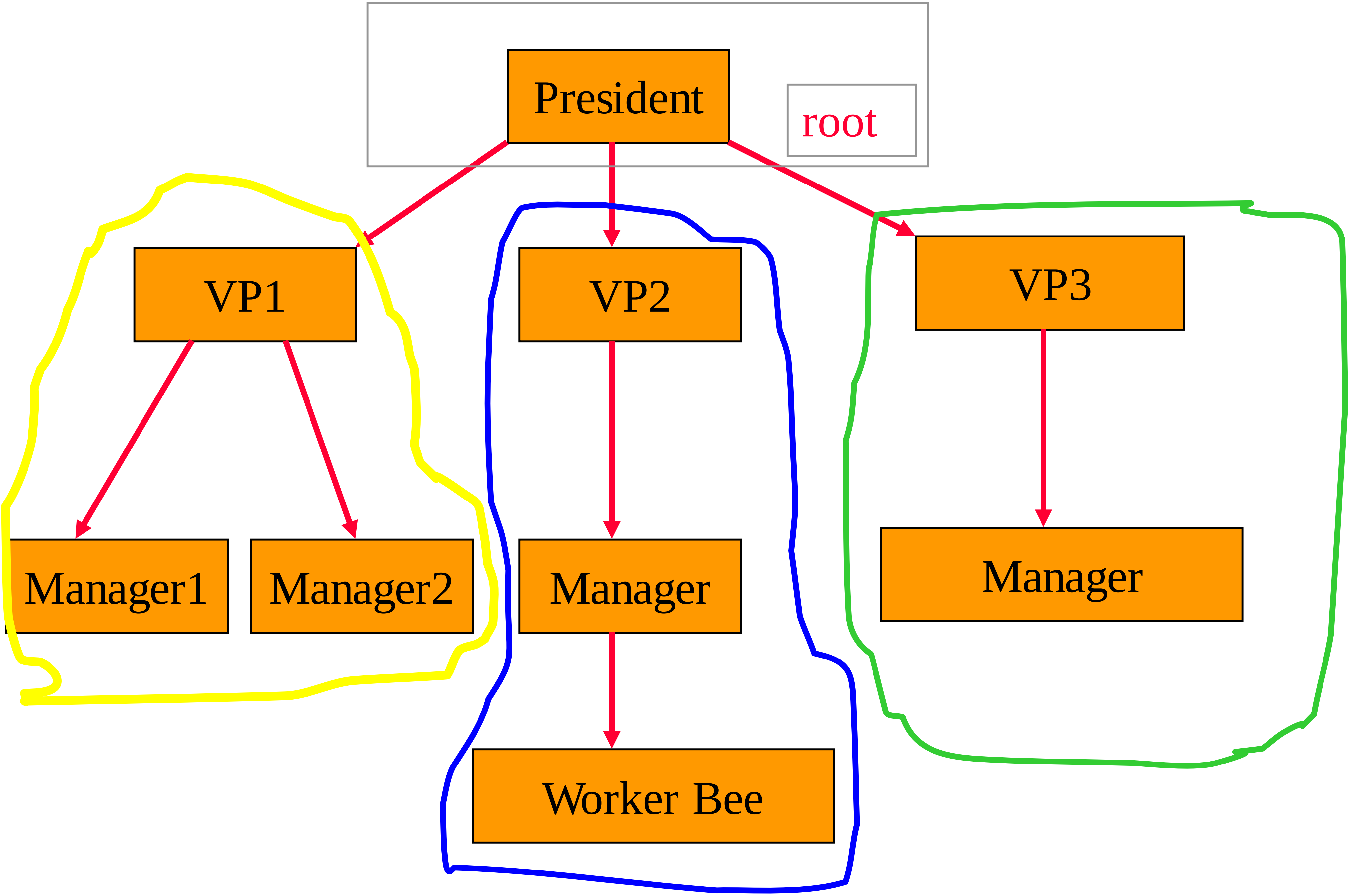
Example Tree- File System



Definition

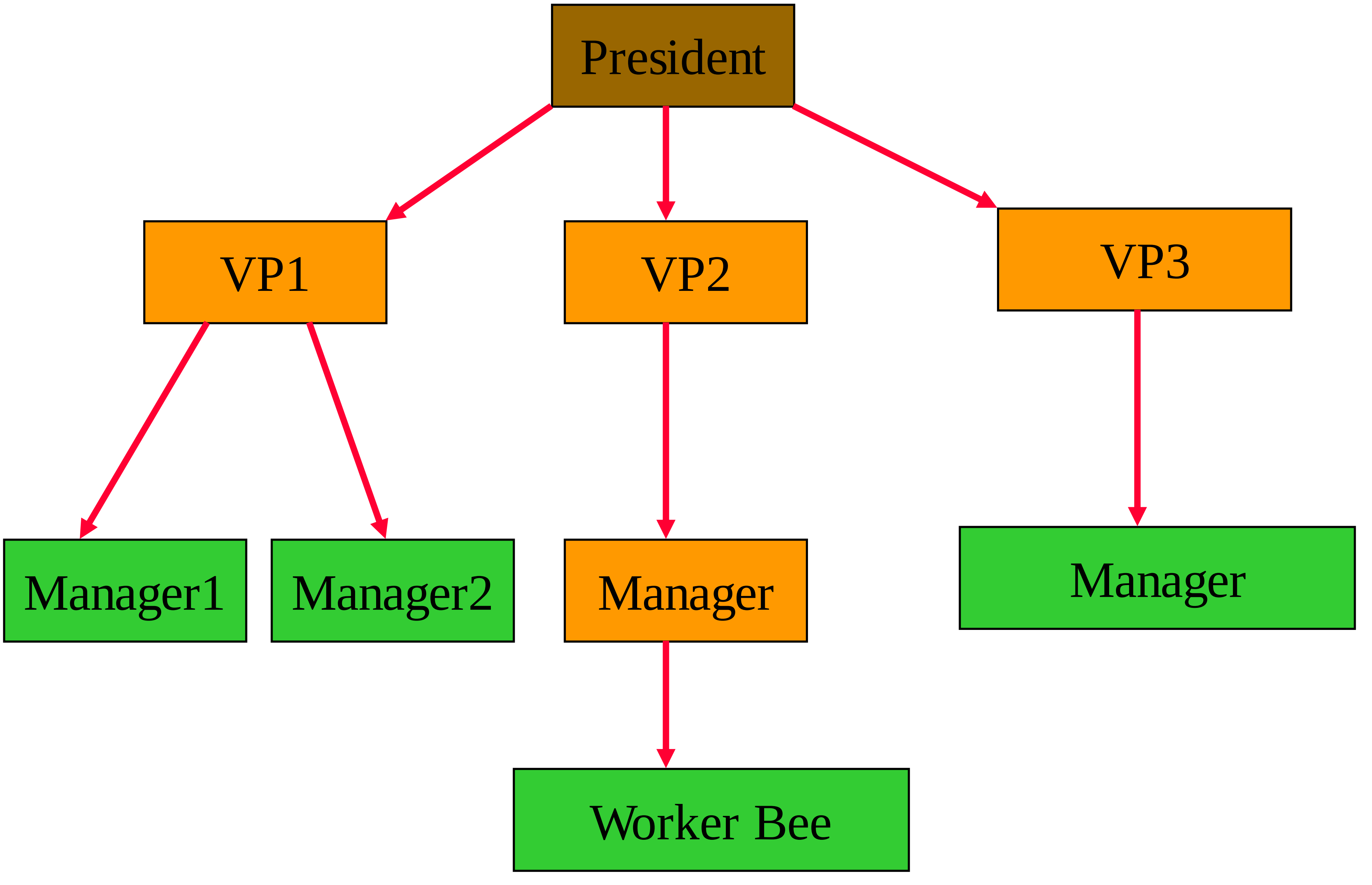
- A tree t is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of t.

Subtrees

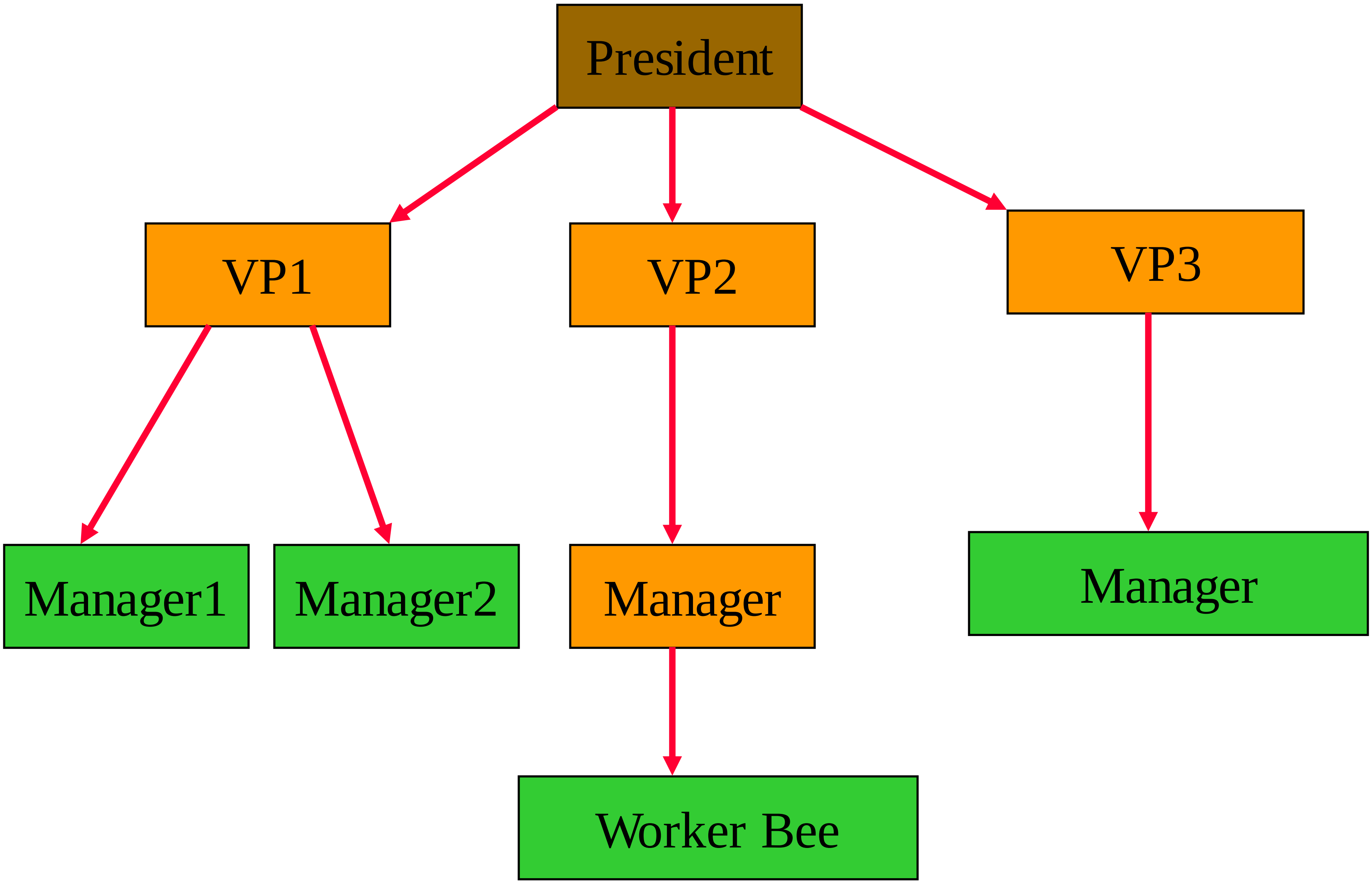


President
root
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

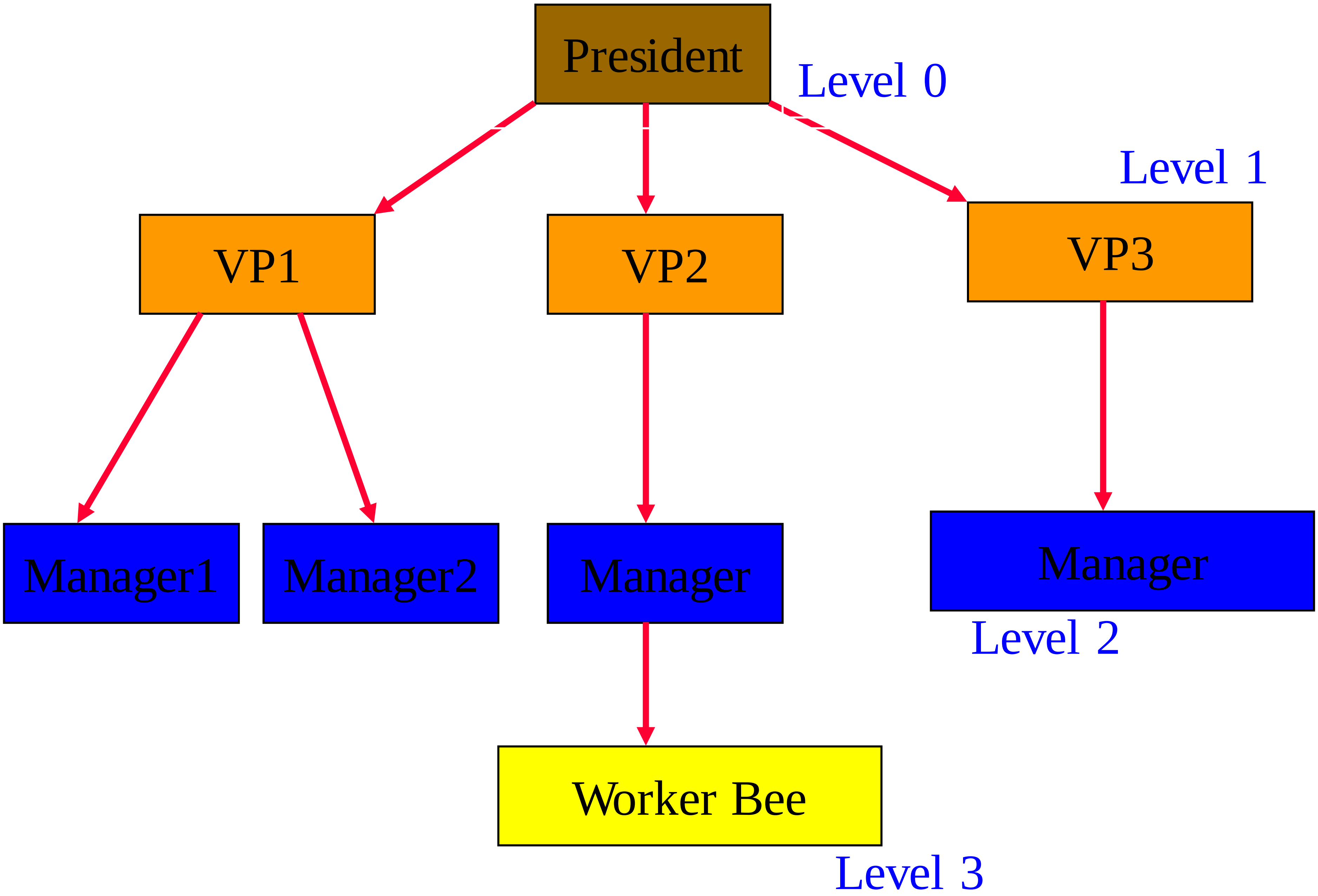
Leaves


President
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

Parent, Grandparent, Siblings, Ancestors, Descendants



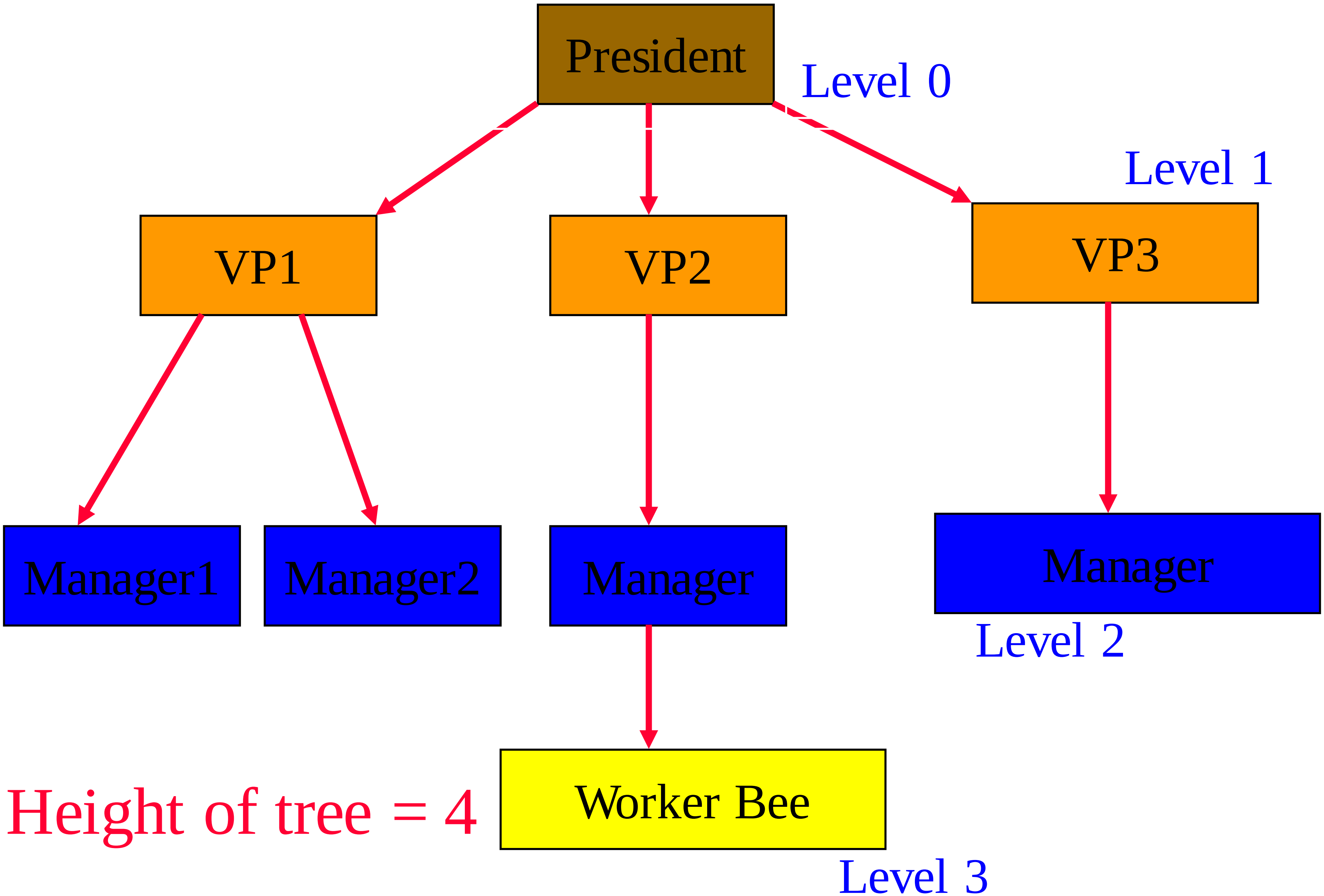
Levels


President
Level 0
Level 1
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Level 2
Worker Bee
Level 3

Caution

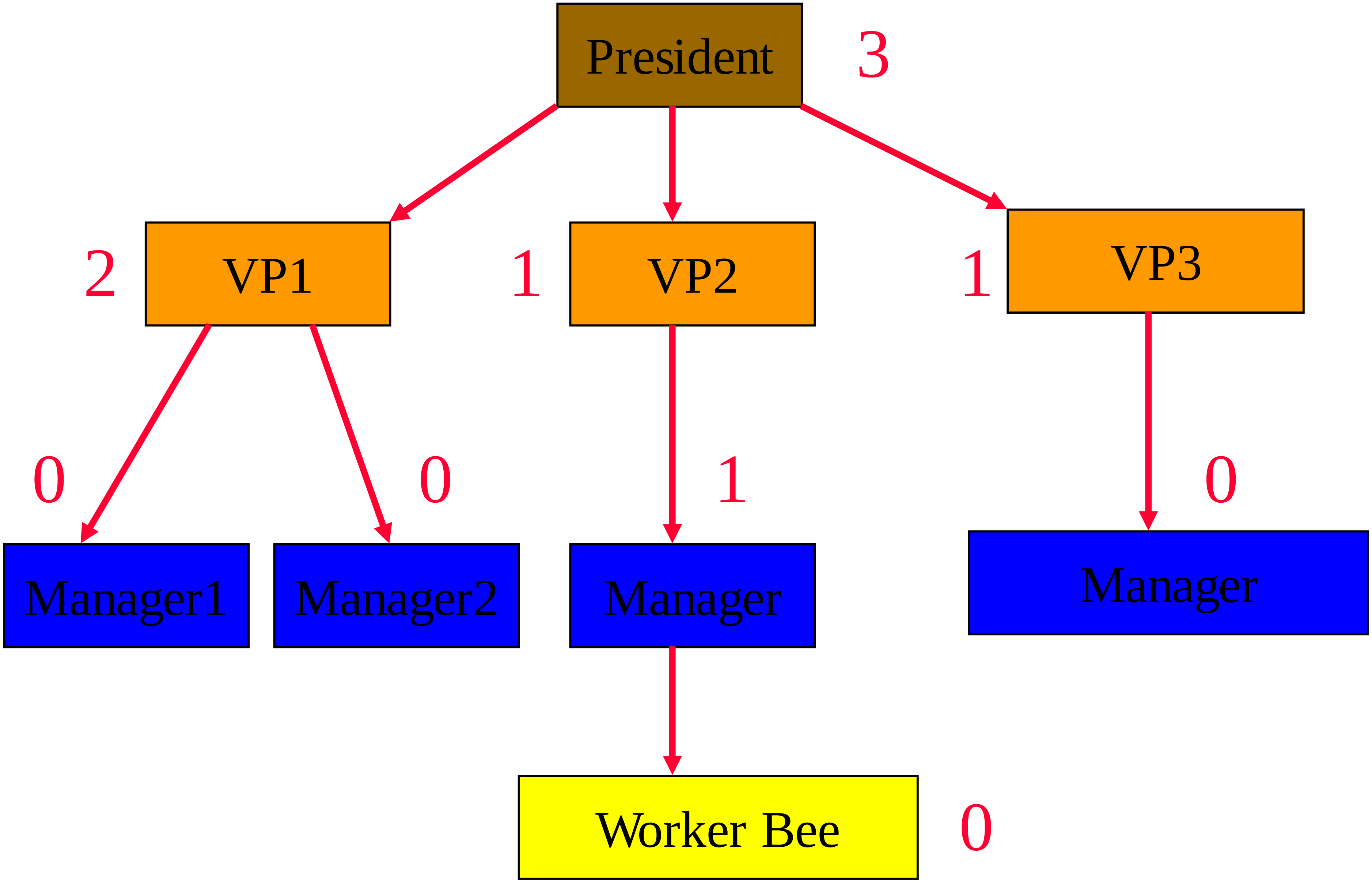
- Some texts start level numbers at 0 rather than at 1.
- Root is at level 0.
- Its children are at level 1.
- The grand children of the root are at level 2.
- And so on.

height = depth = number of levels

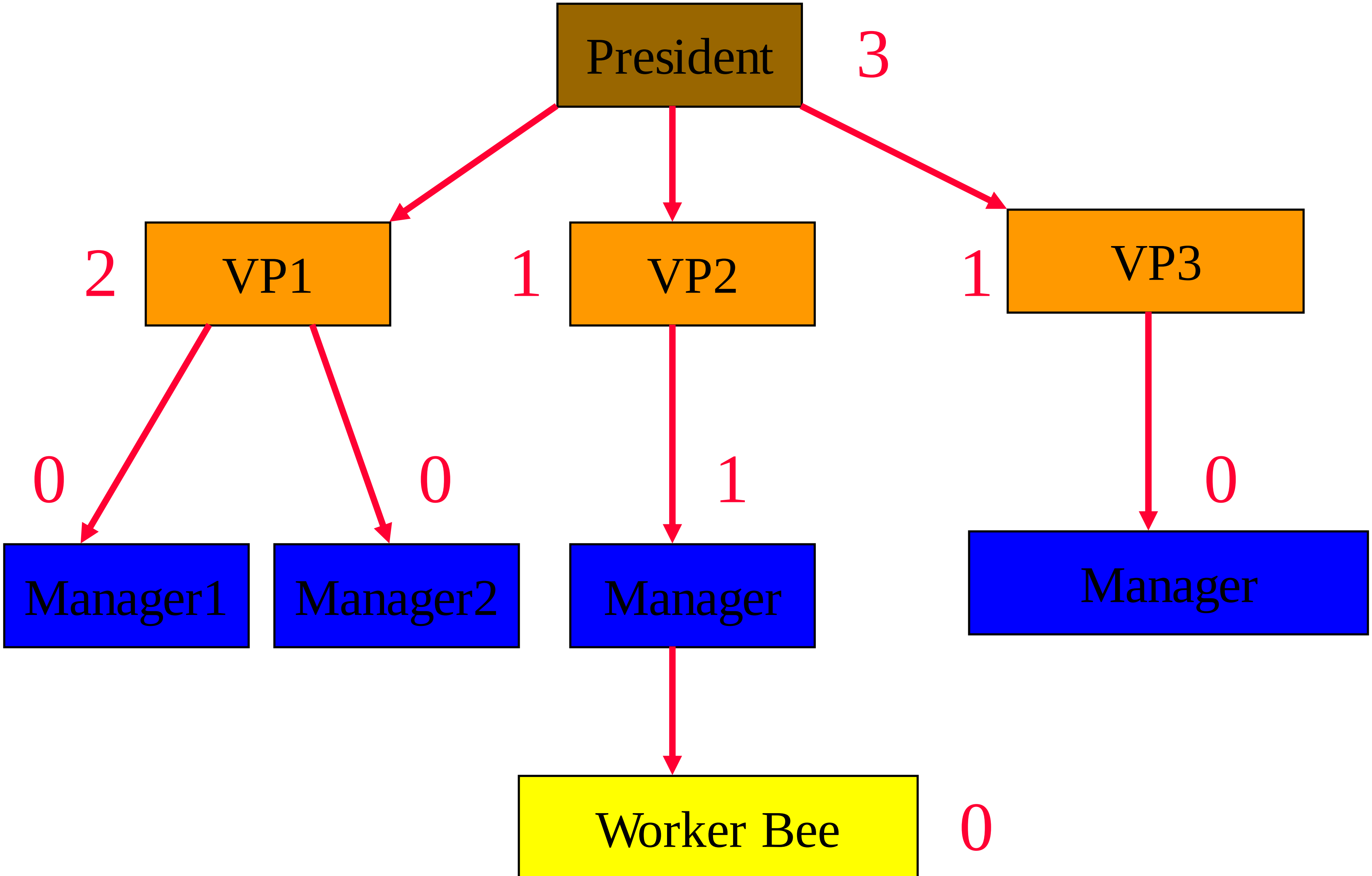


Height of tree = 4

Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

Differences Between A Tree & A Binary Tree

- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- A binary tree may be empty; a tree cannot be empty.

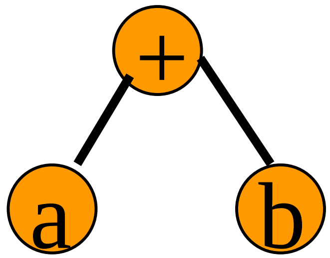
Differences Between A Tree & A Binary Tree

- The subtrees of a binary tree are ordered; those of a tree are not ordered.

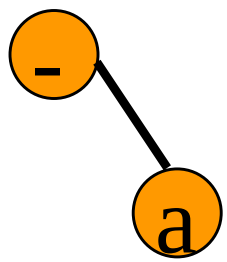
- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Binary Tree Form

- a + b

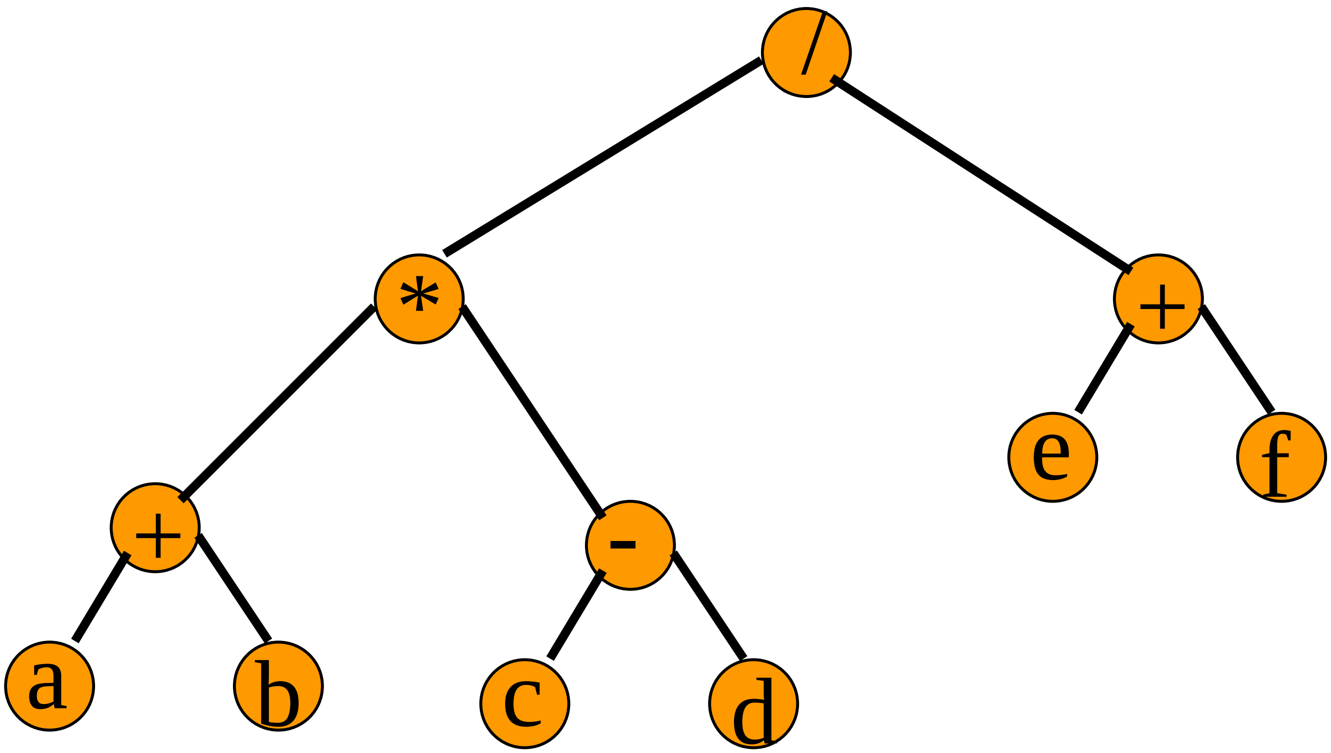


- - a



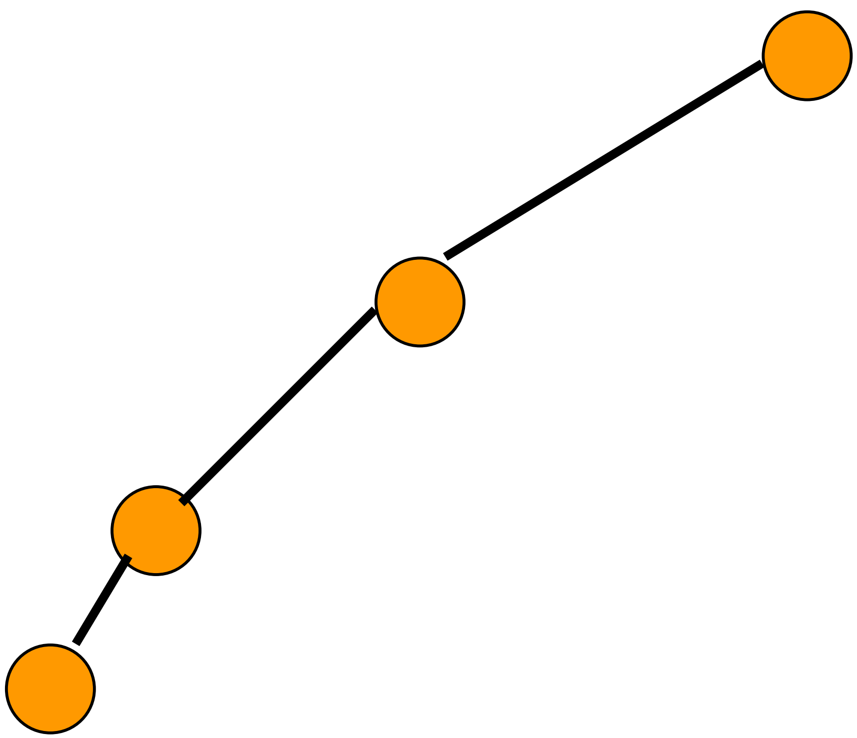
Binary Tree Form

- (a + b) * (c – d) / (e + f)



Minimum Number Of Nodes

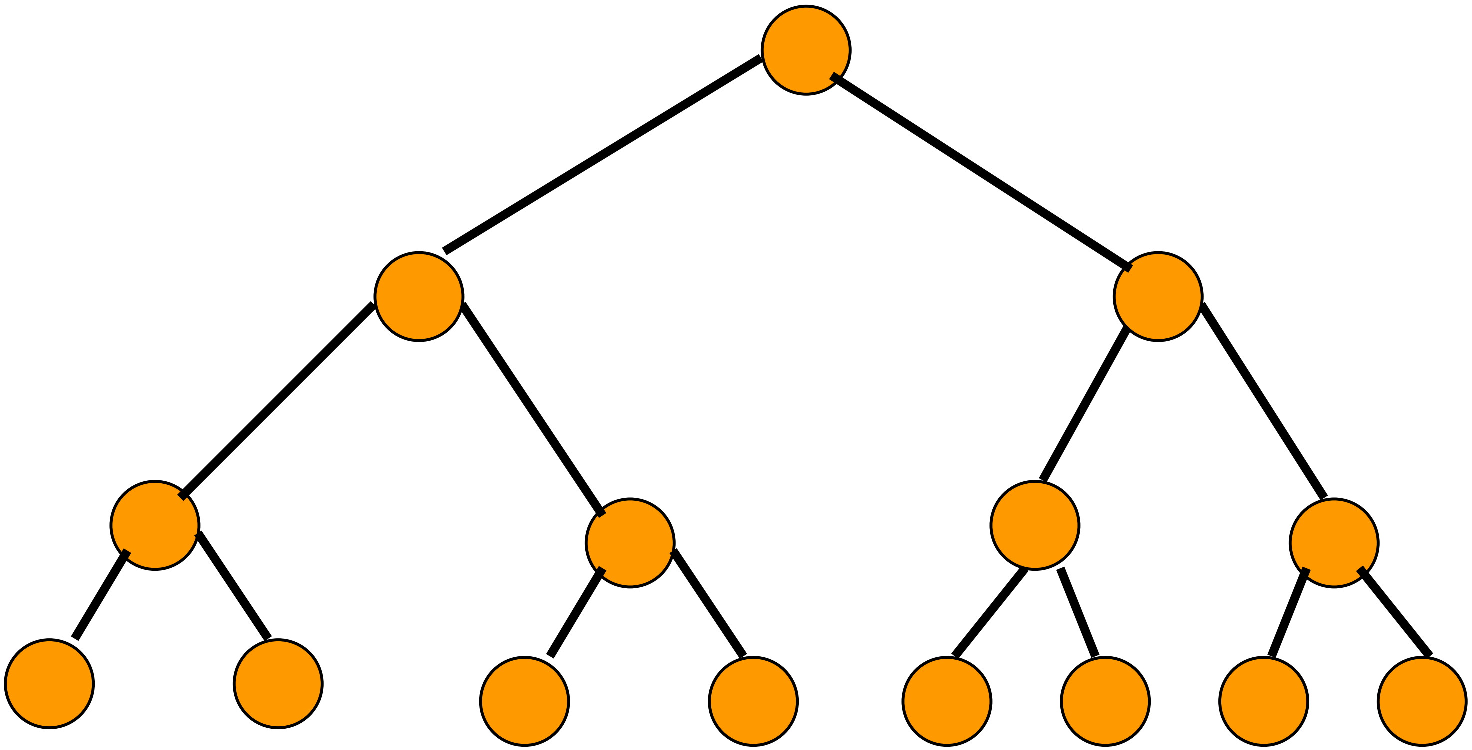
- Minimum number of nodes in a binary tree whose height is h=4.
- At least one node at each of first h levels.



minimum number of nodes is h=4.

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



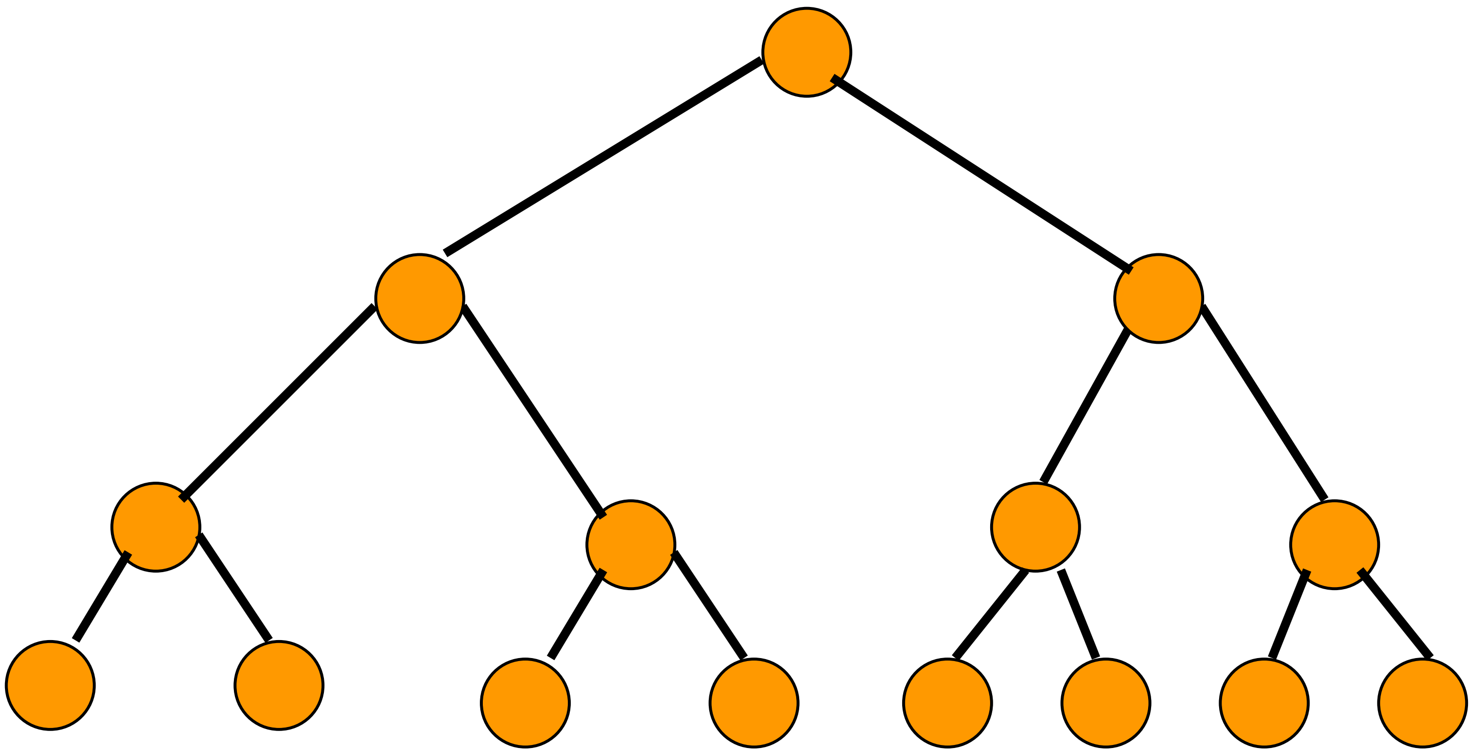
Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

$= 2^h - 1$

Full Binary Tree

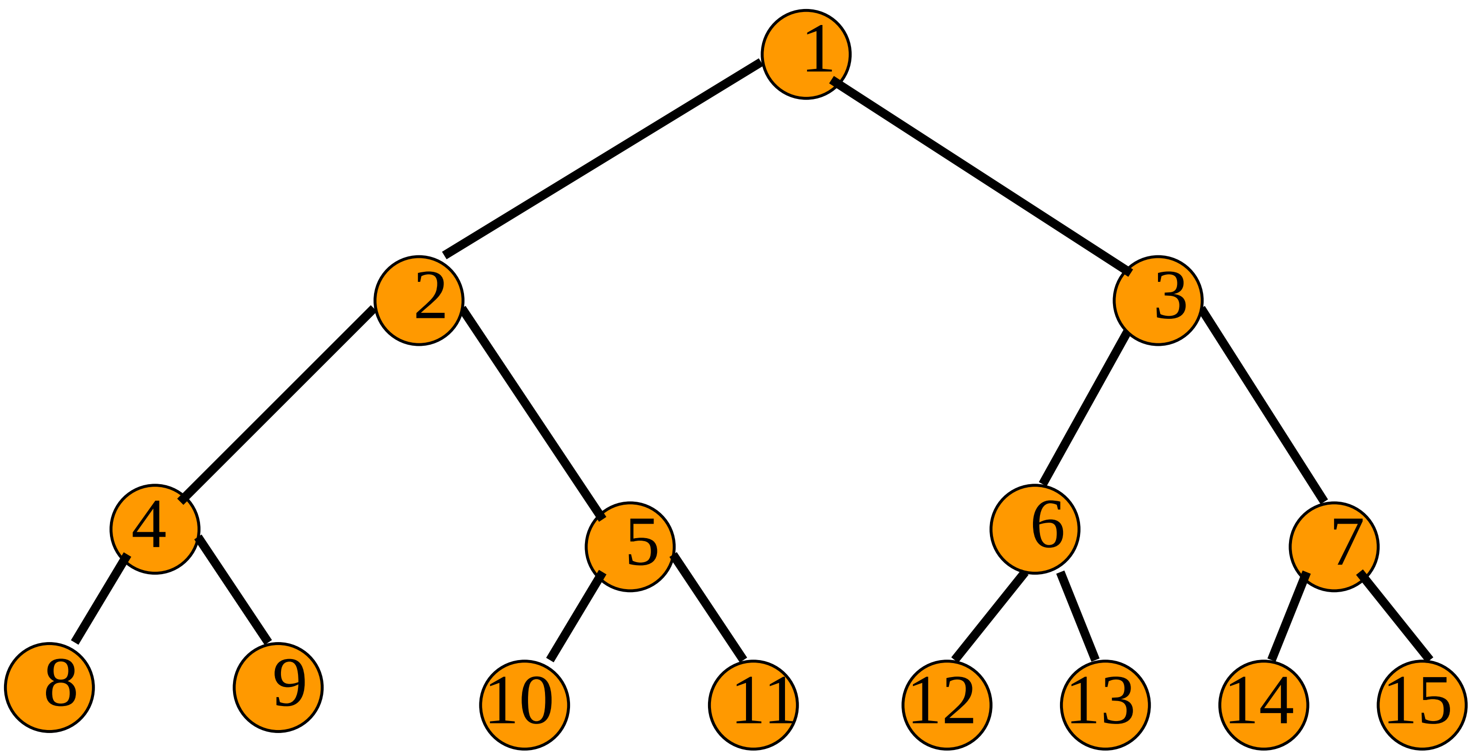
- A full binary tree of a given height h has $2^h - 1$ nodes.



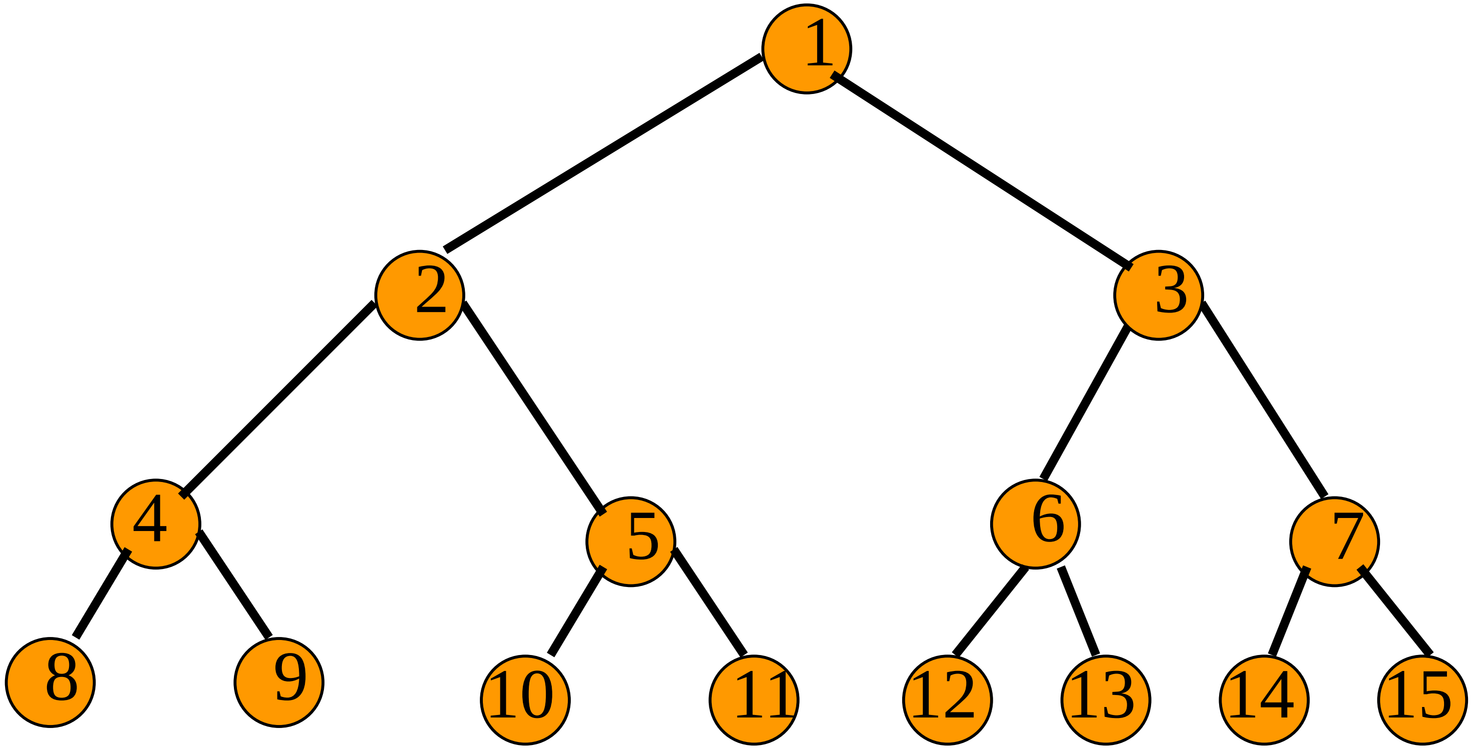
Height 4 full binary tree.

Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.

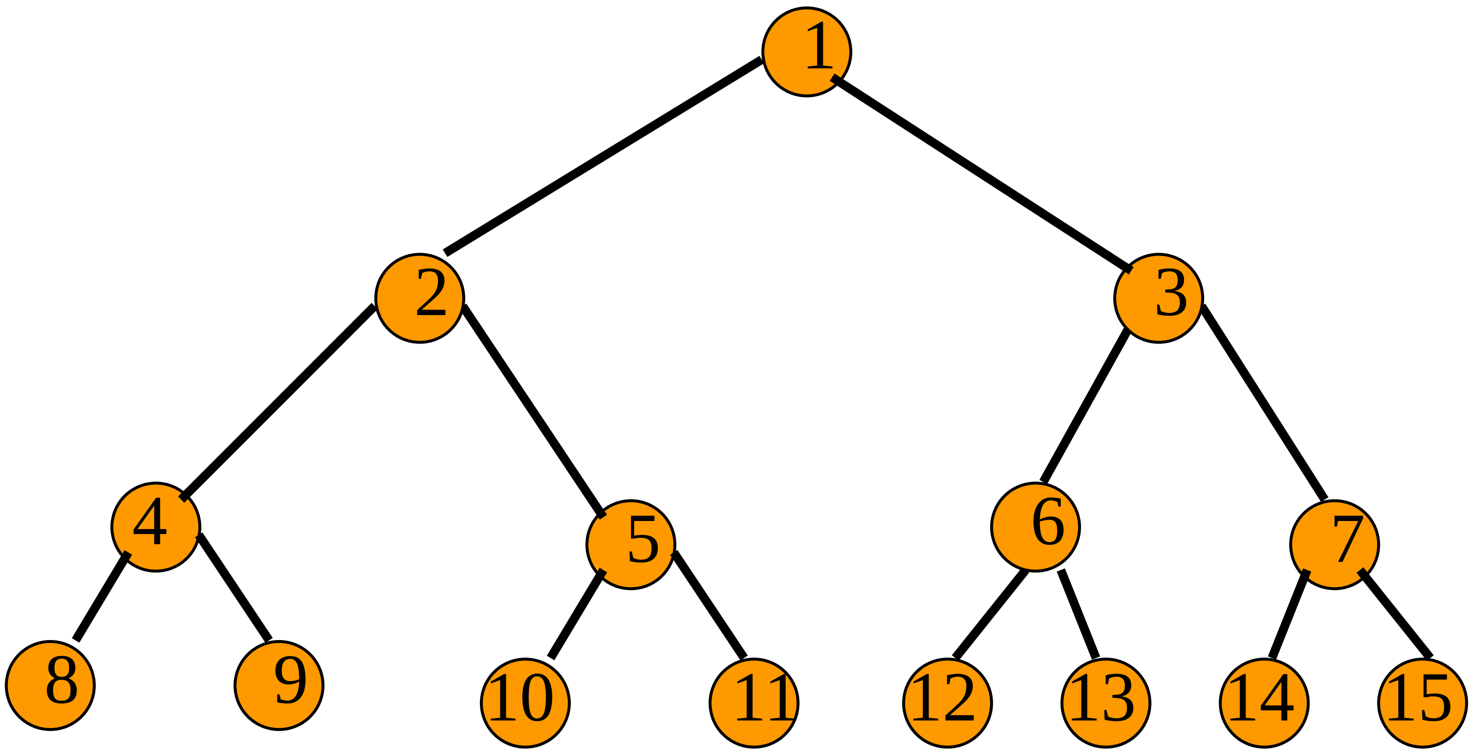


Node Number Properties



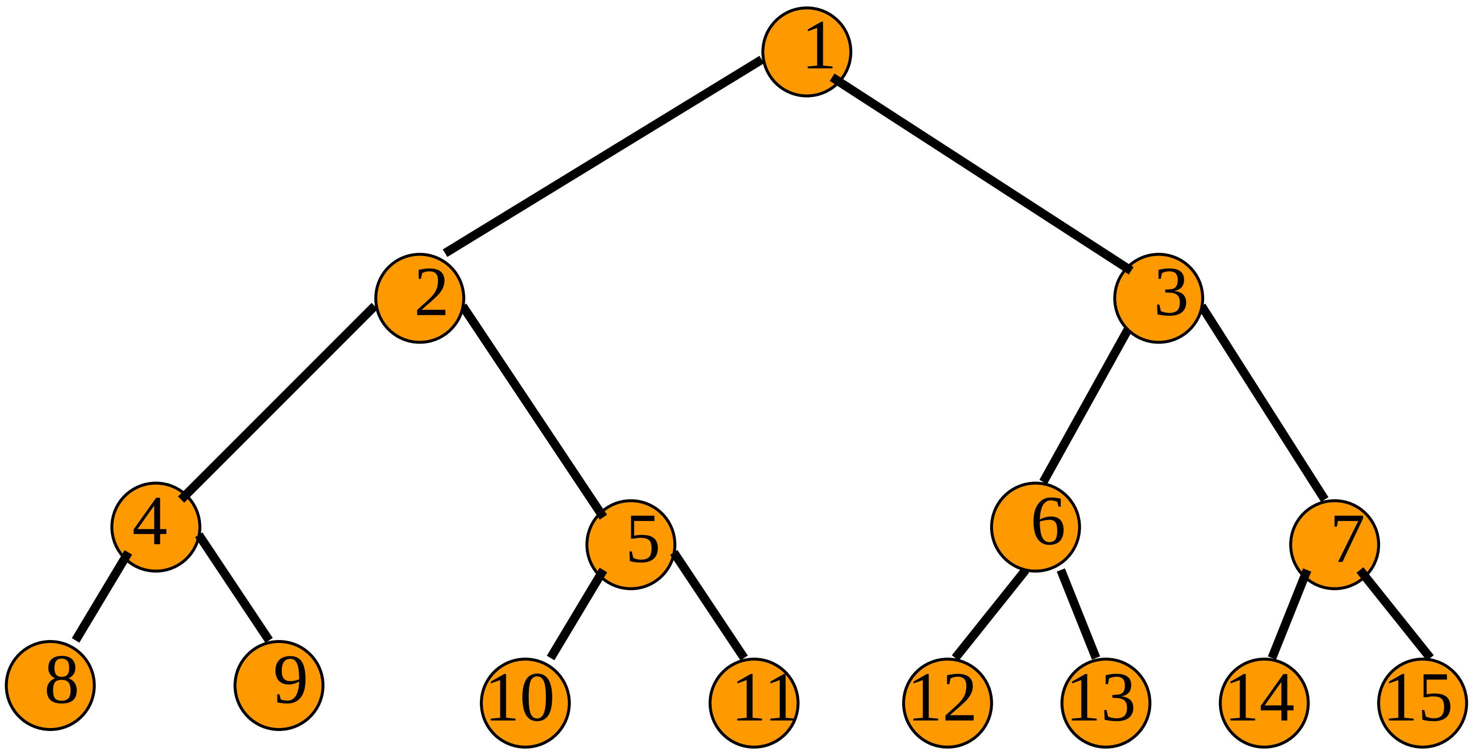
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

Node Number Properties



- Left child of node i is node $2i$, unless $2i > n$, where n is the number of nodes.
- If $2i > n$, node i has no left child.

Node Number Properties

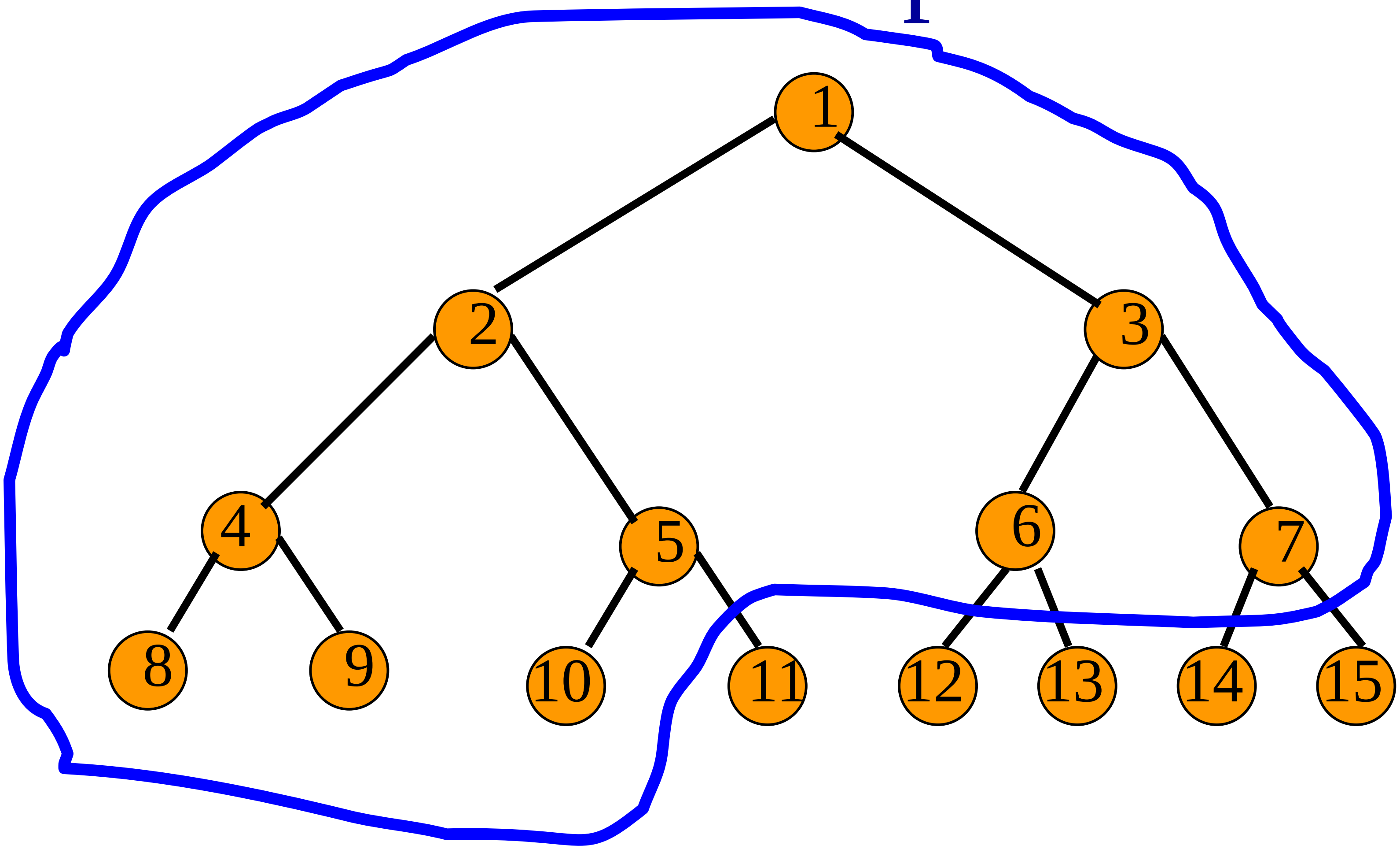


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i+1 > n, node i has no right child.

Complete Binary Tree With n Nodes

- Start with a full binary tree that has at least n nodes.
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree.

Example



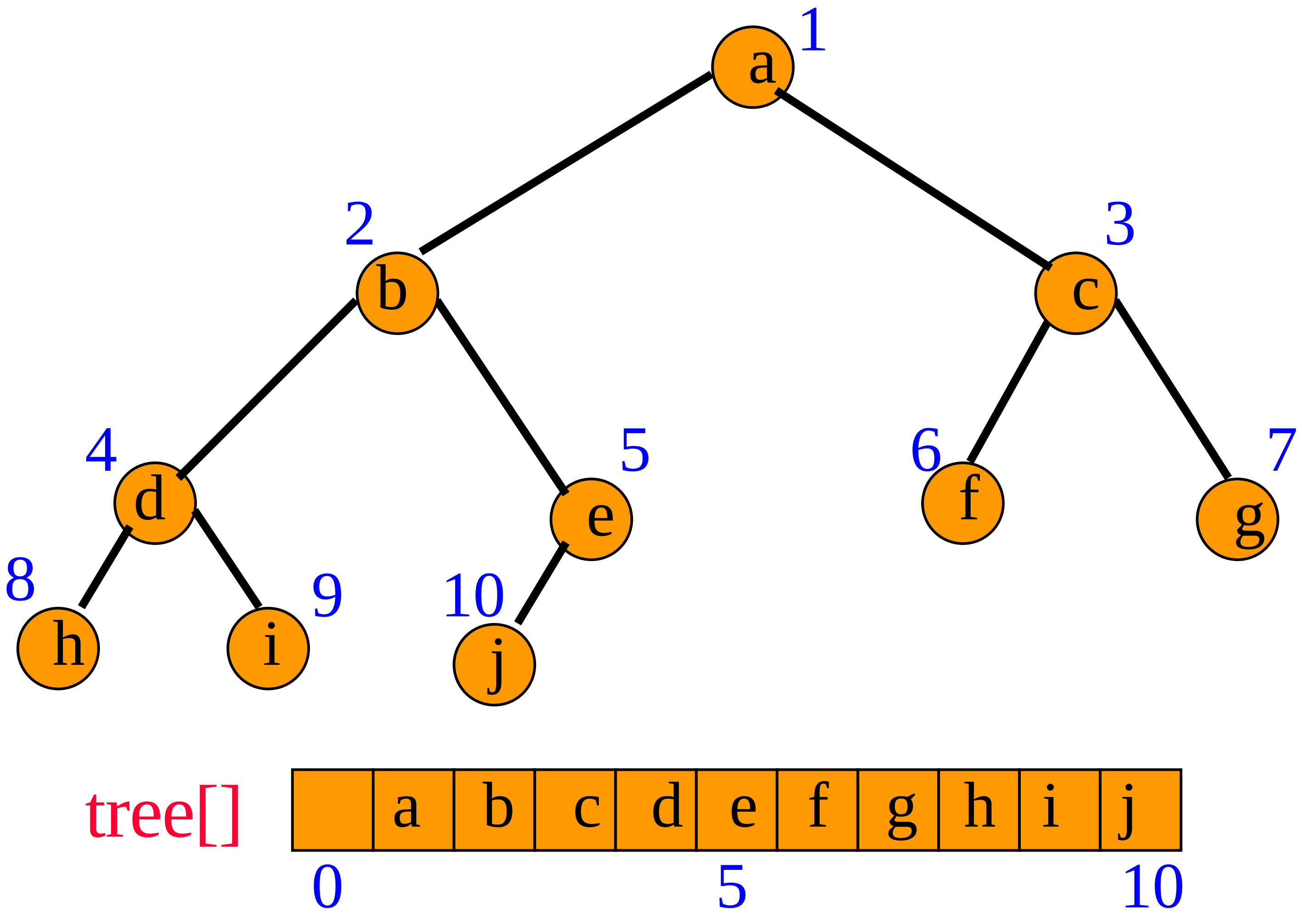
- Complete binary tree with 10 nodes.

Binary Tree Representation

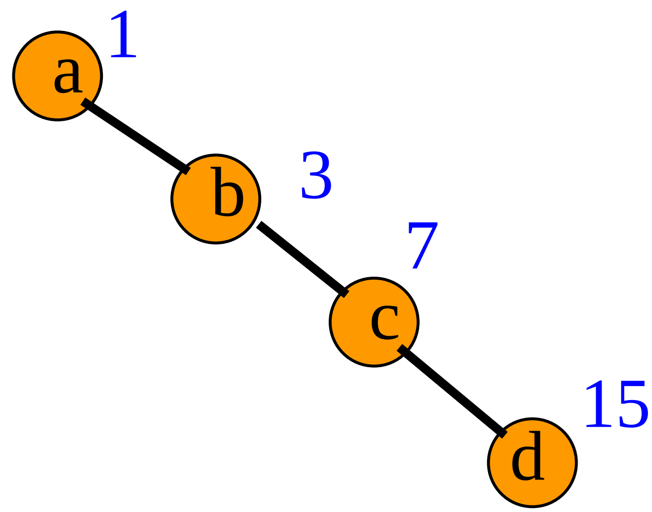
- Array representation.
- Linked representation.

Array Representation

- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].



Right-Skewed Binary Tree



tree[]		a	-	b	-	-	-	c	-	-	-	-	-	-	-	d
	0					5					10					15

- An n node binary tree needs an array whose length is between $n+1$ and 2^n.

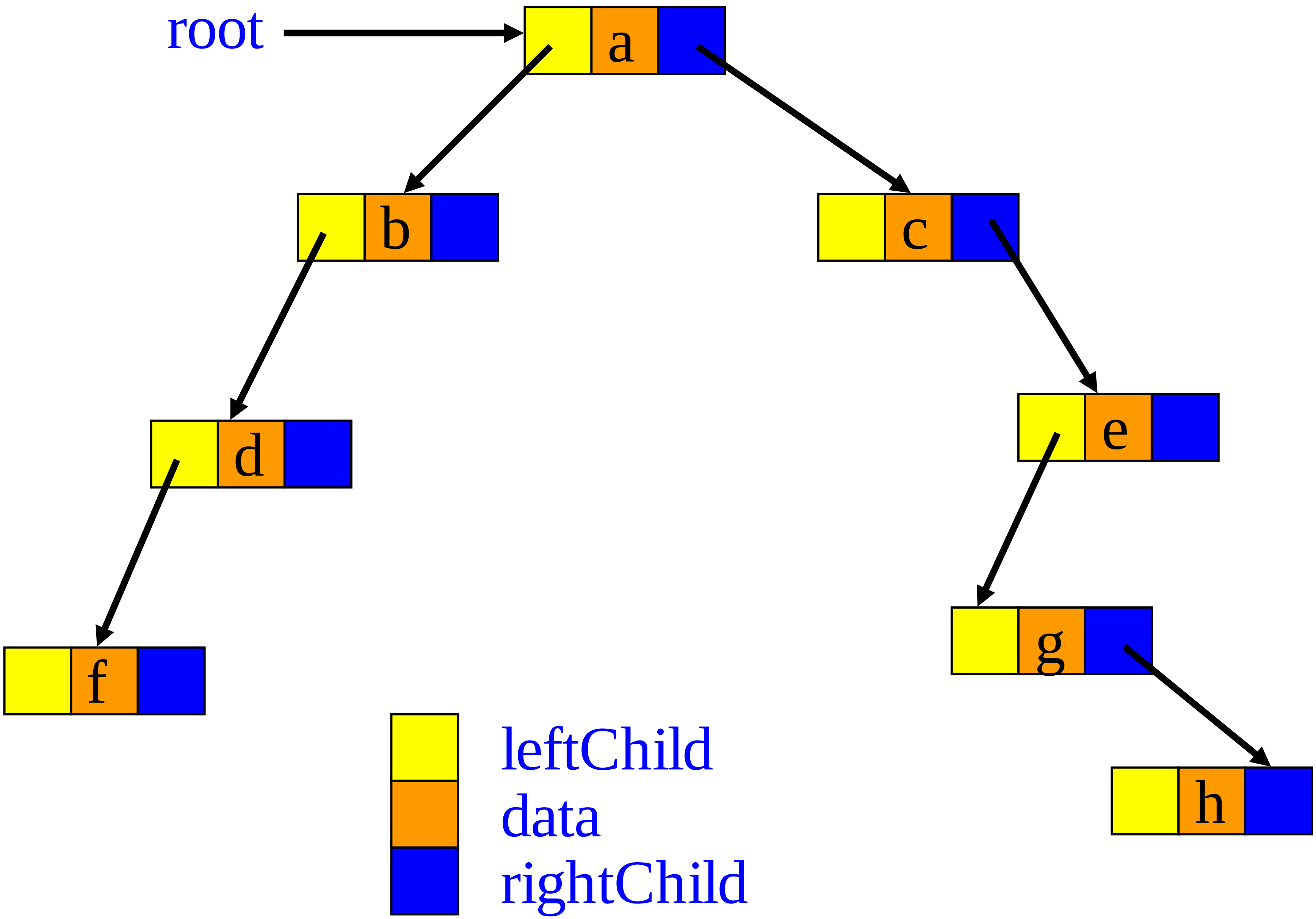
Linked Representation

- Each binary tree node is represented as an object whose data type is Tree Node.
- The space required by an n node binary tree is n * (space required by one node).

Node Representation

```
typedef struct node *treePointer;
typedef struct {
        char data;
        treePointer leftChild, rightChild;
        } node;
```

Linked Representation Example



Some Binary Tree Operations

- Determine the height.
- Determine the number of nodes.
- Make a clone.
- Determine if two binary trees are clones.
- Display the binary tree.
- Evaluate the arithmetic expression represented by a binary tree.
- Obtain the infix form of an expression.
- Obtain the prefix form of an expression.
- Obtain the postfix form of an expression.

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree.
- In a traversal, each element of the binary tree is visited exactly once.
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

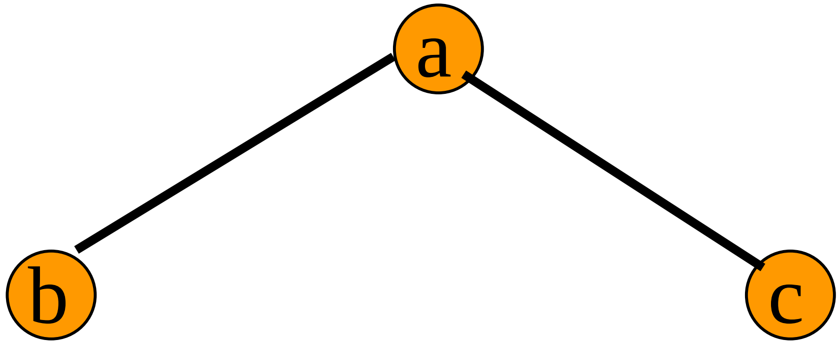
Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder

Preorder Traversal

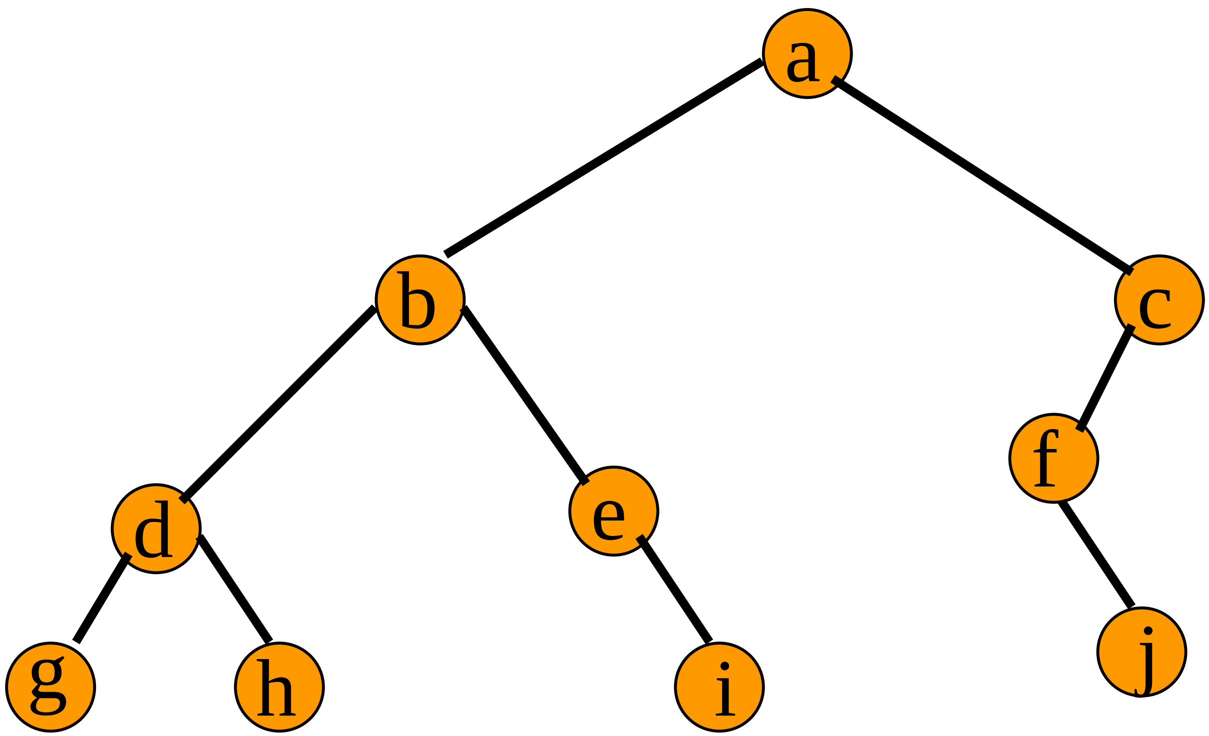
```
void preOrder(treePointer ptr)
{
    if (ptr != NULL)
    {
        visit(t);
        preOrder(ptr->leftChild);
        preOrder(ptr->rightChild);
    }
}
```

Preorder Example (Visit = print)



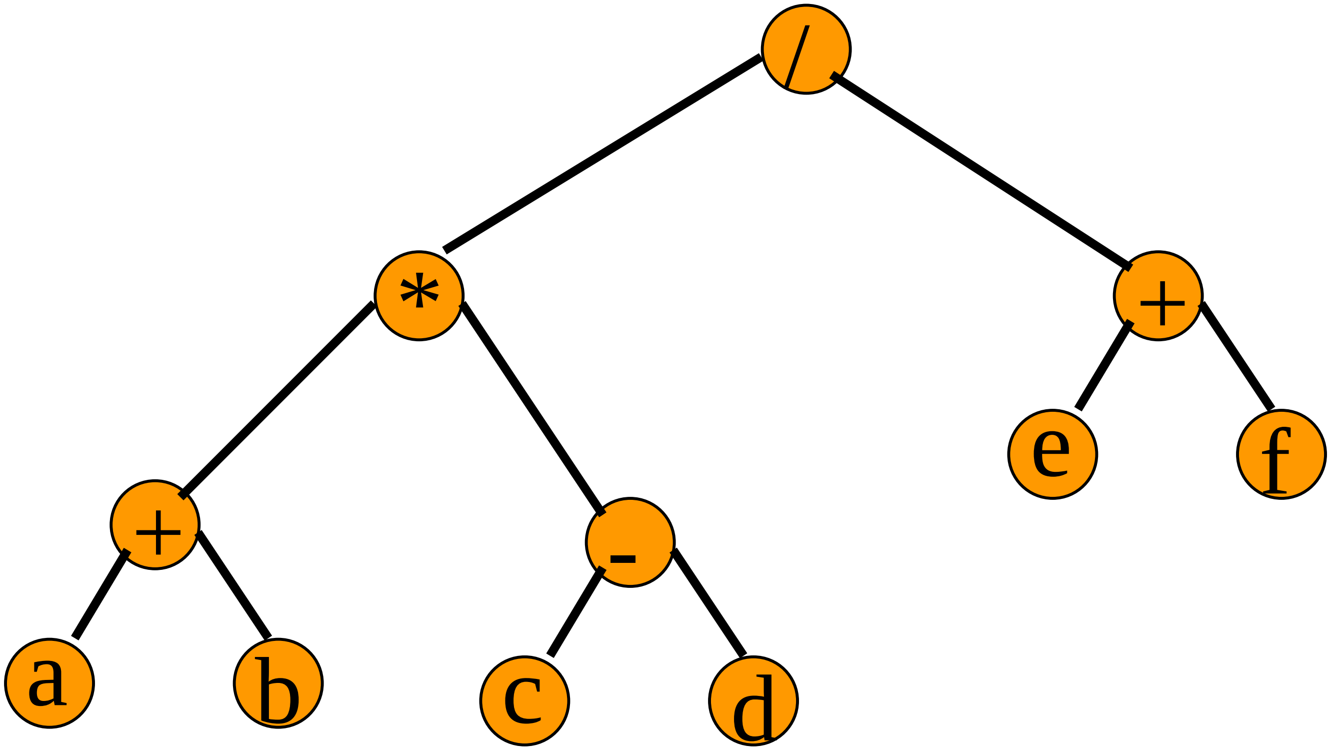
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

Preorder Of Expression Tree

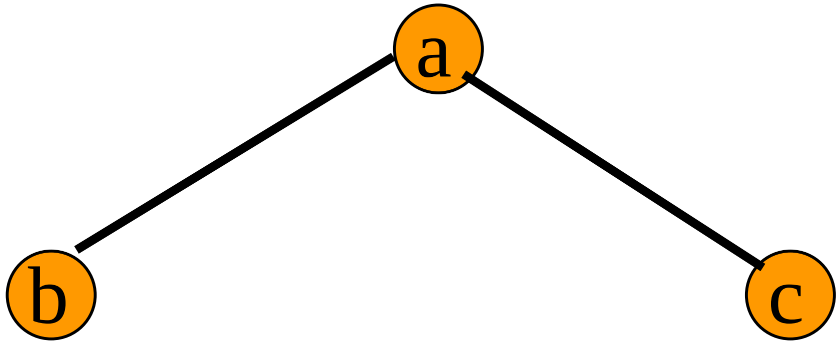


/ * + a b - c d + e f

Inorder Traversal

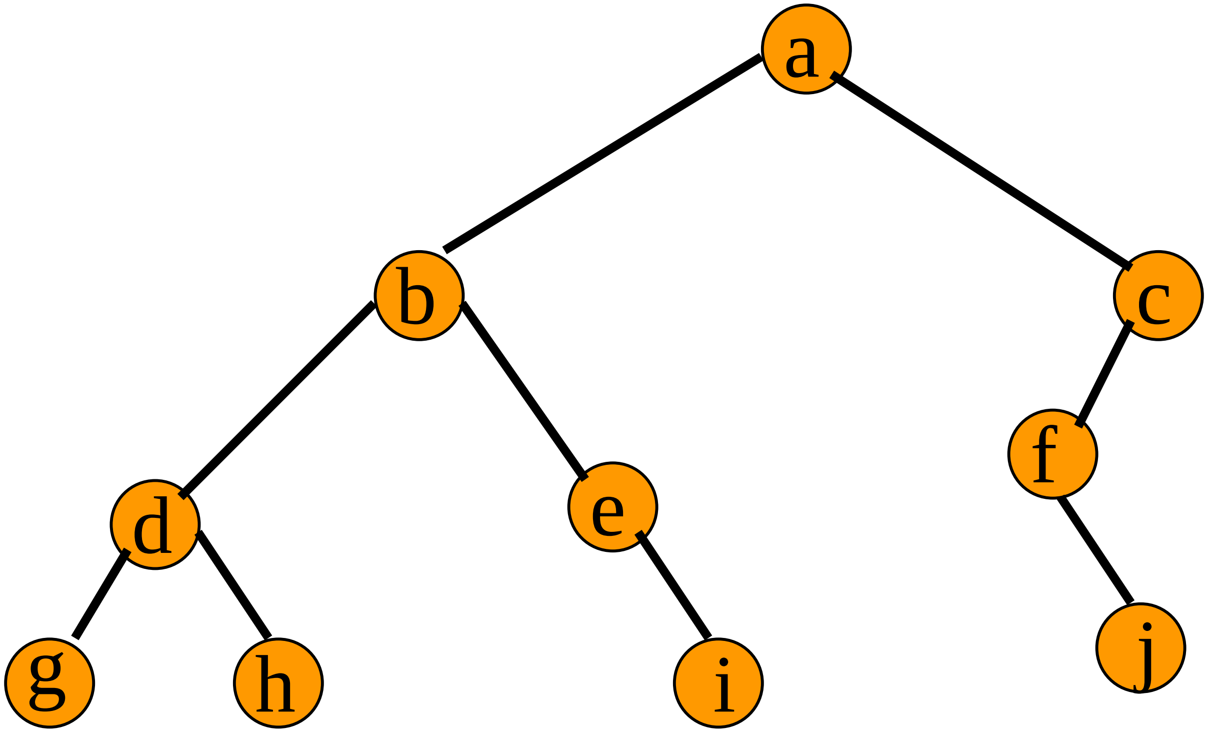
```
void inOrder(treePointer ptr)
{
    if (ptr != NULL)
    {
        inOrder(ptr->leftChild);
        visit(ptr);
        inOrder(ptr->rightChild);
    }
}
```

Inorder Example (Visit = print)



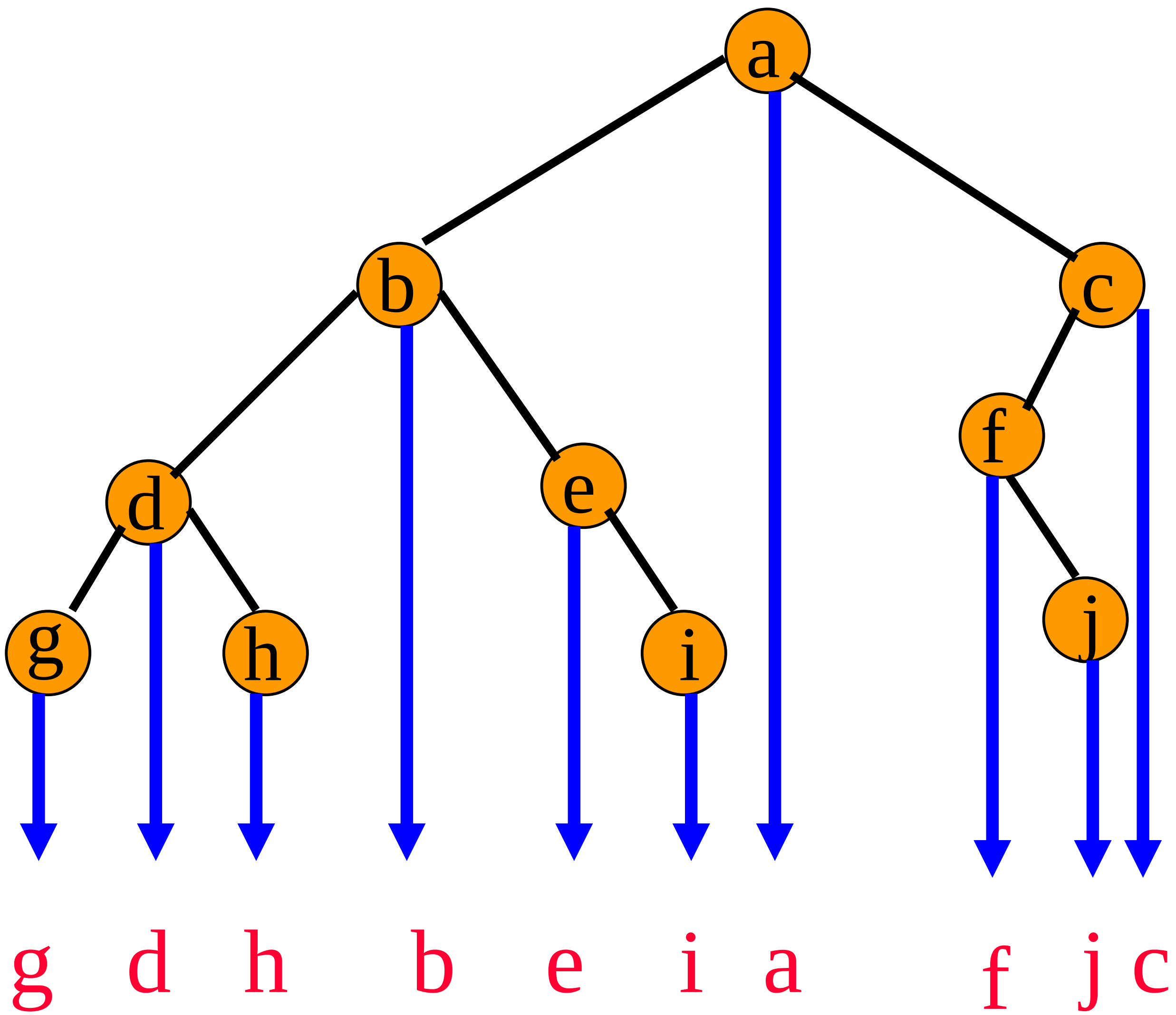
b a c

Inorder Example (Visit = print)

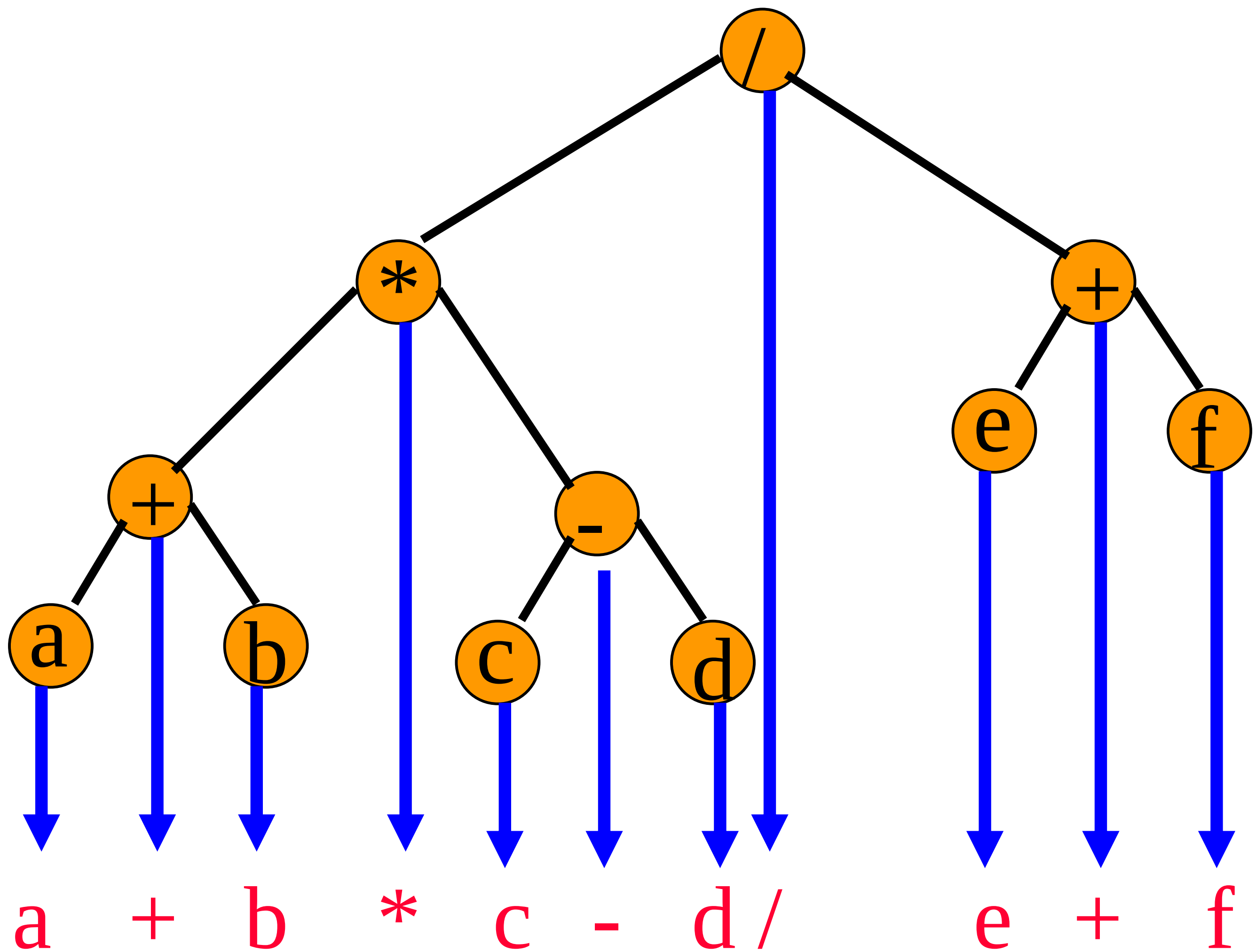


g d h b e i a f j c

Inorder By Projection (Squishing)



Inorder Of Expression Tree



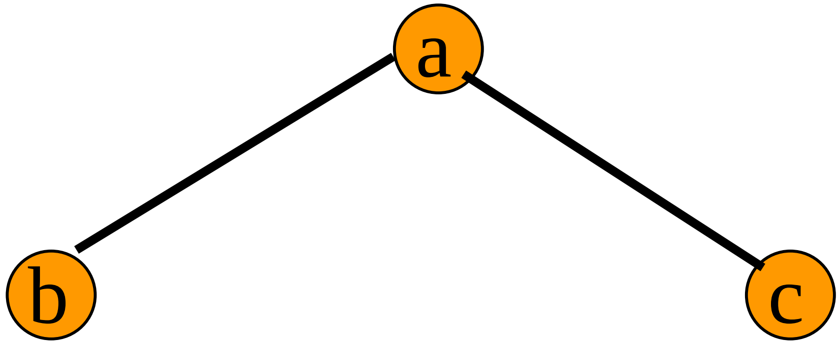
Question

Suppose the numbers **7, 5, 1, 8, 3, 6, 0, 9, 4, 2** are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?

Postorder Traversal

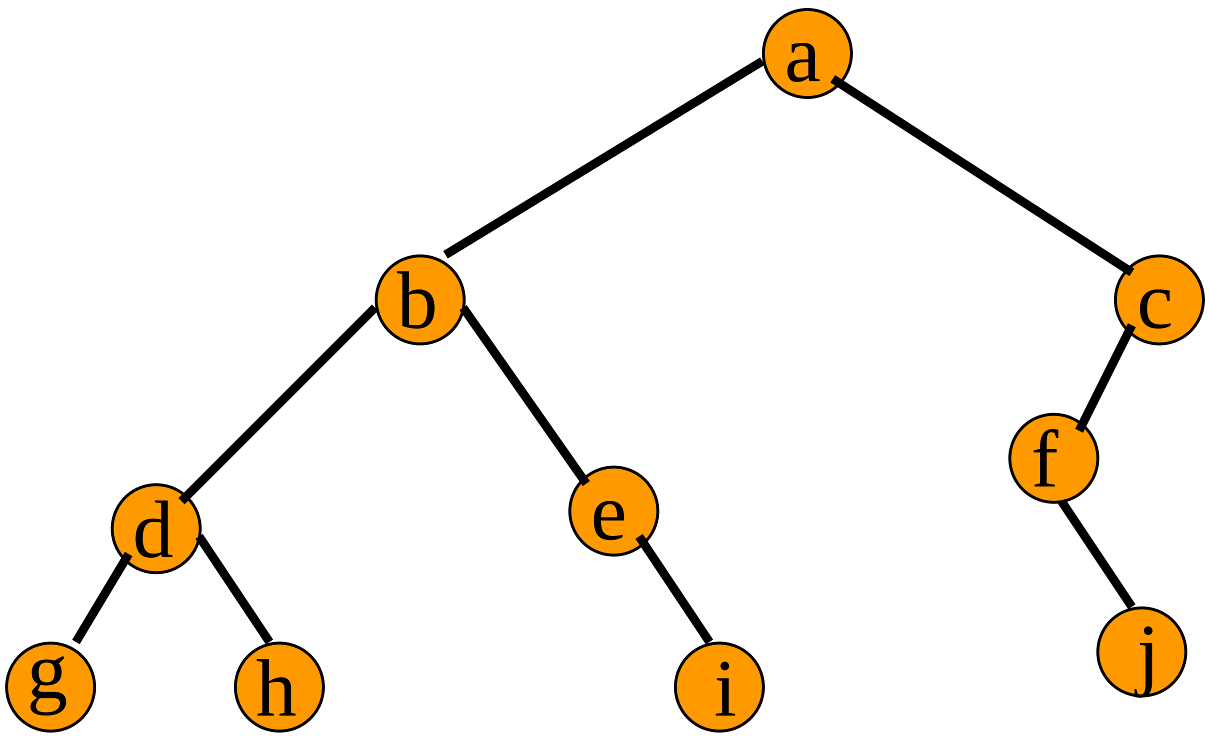
```
void postOrder(treePointer ptr)
{
    if (ptr != NULL)
    {
        postOrder(ptr->leftChild);
        postOrder(ptr->rightChild);
        visit(t);
    }
}
```

Postorder Example (Visit = print)



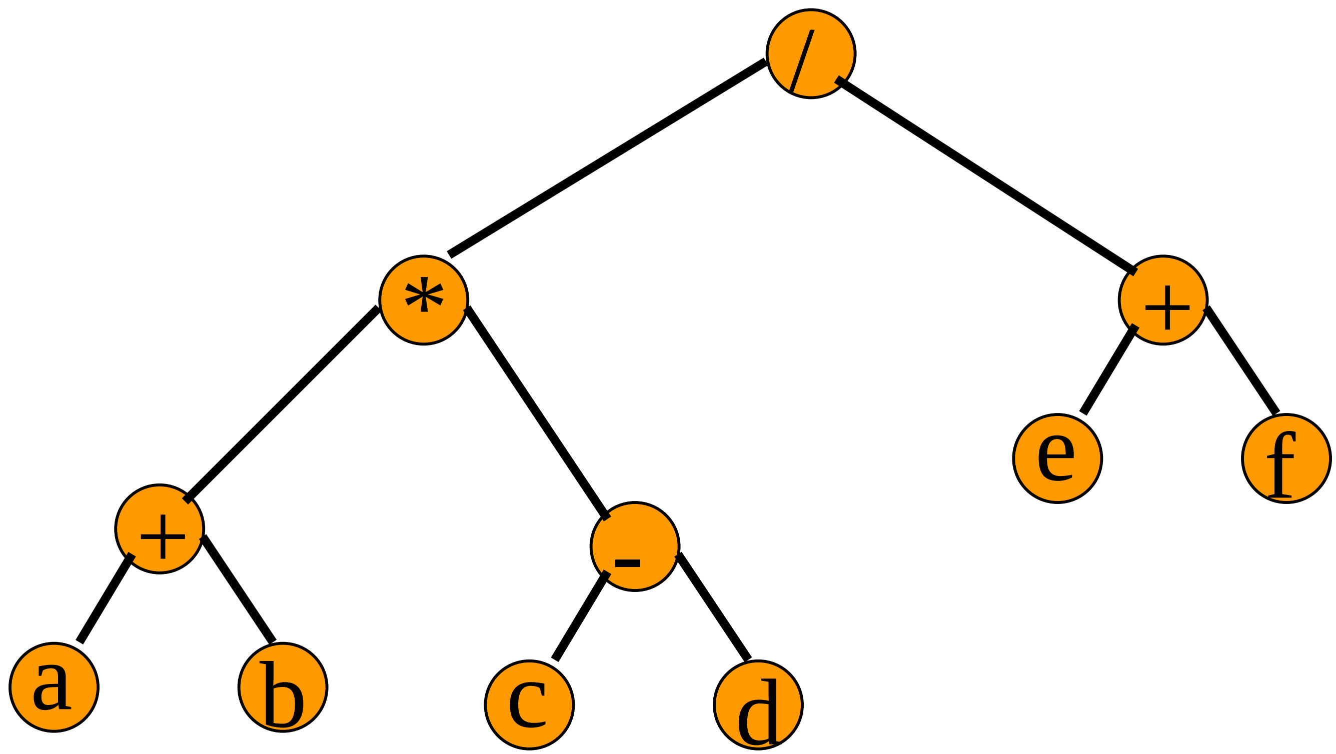
b c a

Postorder Example (Visit = print)



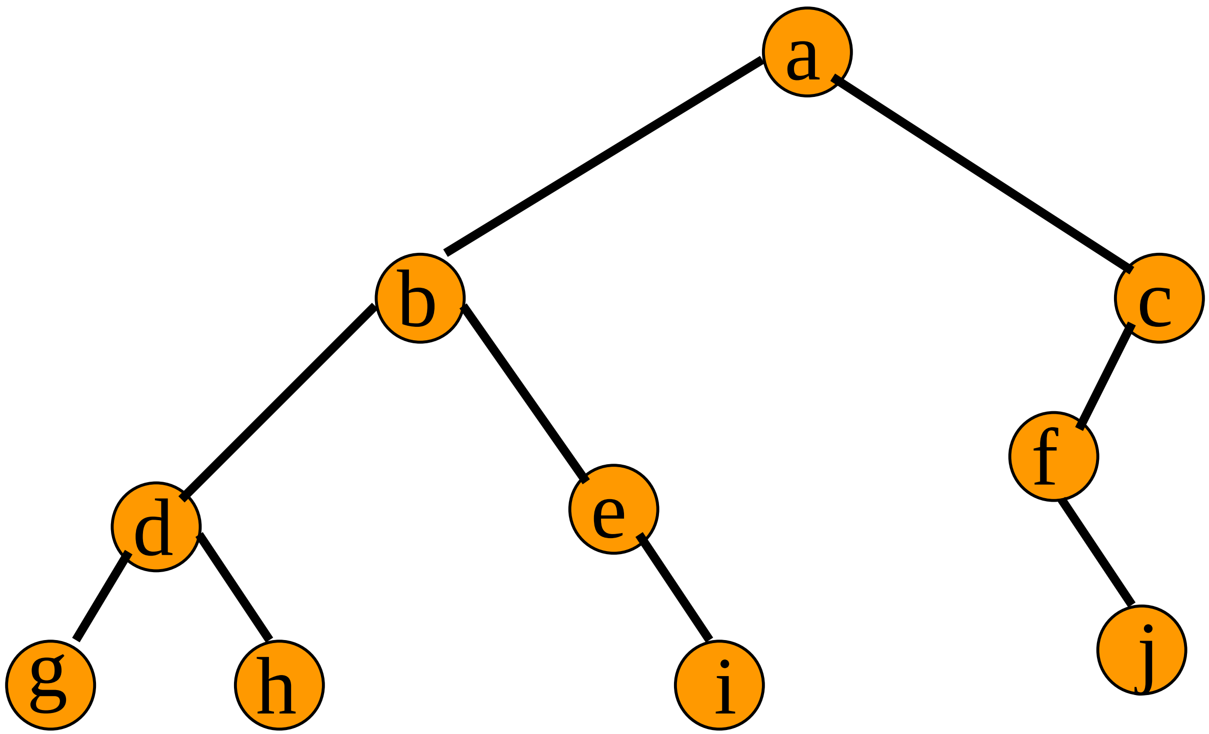
g h d i e b j f c a

Postorder Of Expression Tree



a b + c d - * e f + /

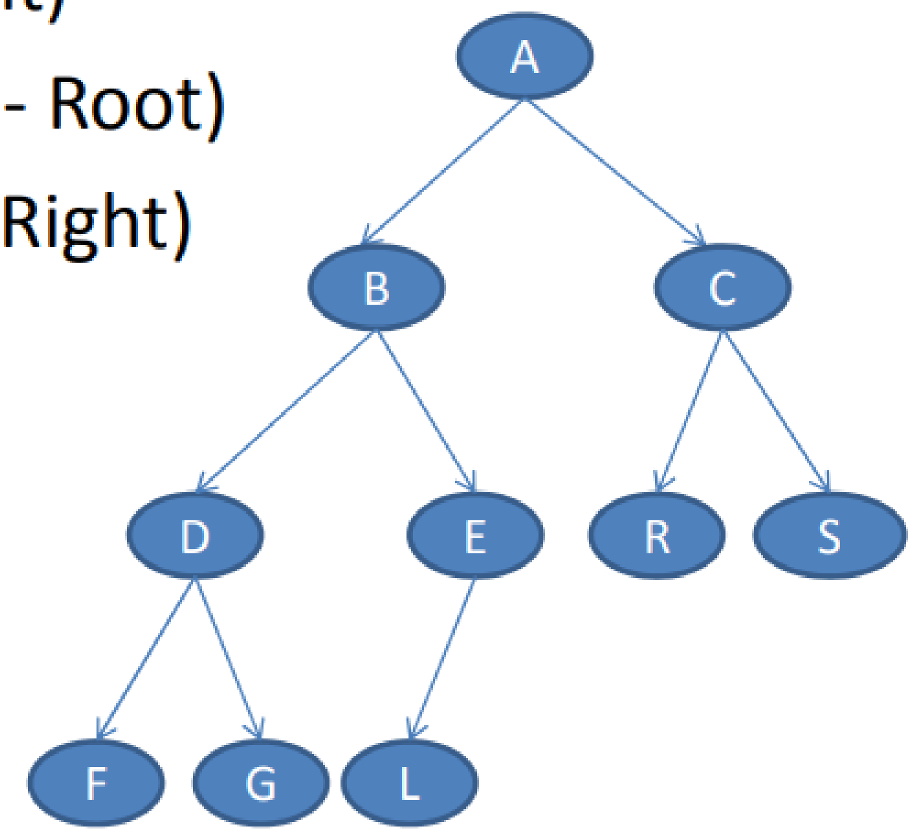
Traversal Applications



- Make a clone.
- Determine height.
- Determine number of nodes.

Example-1

- Perform the other traversals of that tree:
  - inOrder (Left-Root-Right)
  - postOrder (Left - Right - Root)
  - preOrder (Root - Left - Right)



Solution

- In: F, D, G, B, L, E, A, R, C, S
- Post: F, G, D, L, E, B, R, S, C, A
- Pre: A, B, D, F, G, E, L, C, R, S

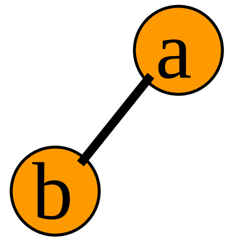
Binary Tree Construction

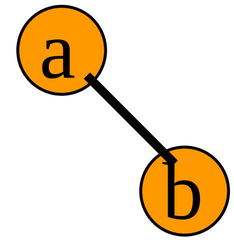
- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely.

Some Examples

preorder

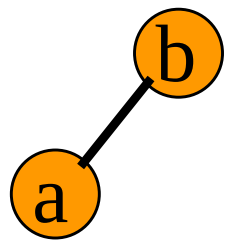
= ab

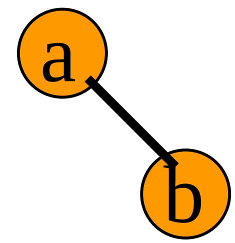




inorder

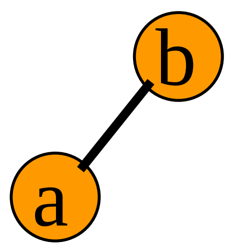
= ab

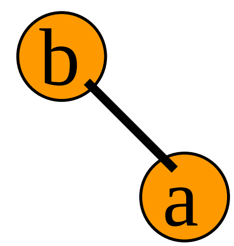




postorder

= ab





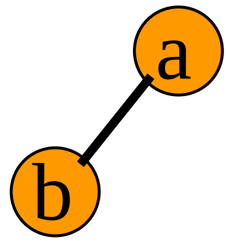
Binary Tree Construction

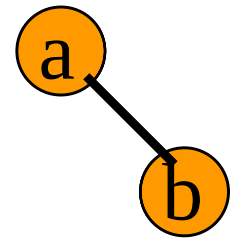
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder And Postorder

preorder = ab

postorder = ba

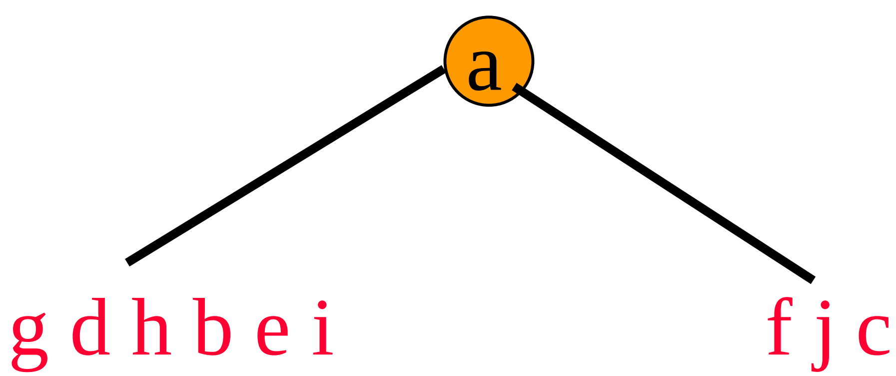




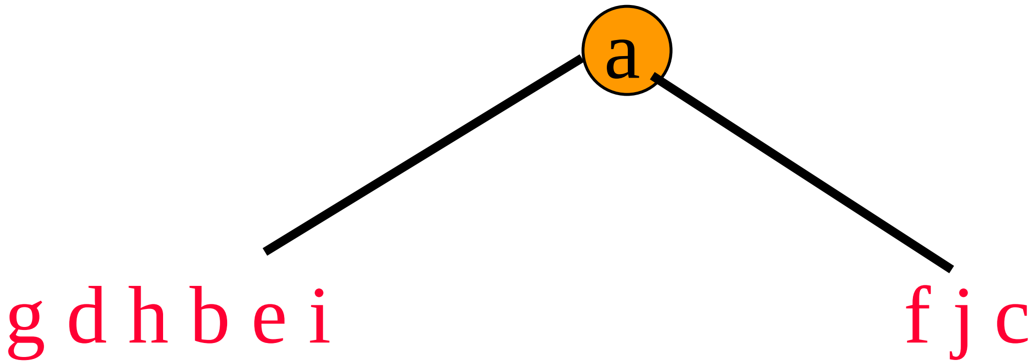
- Preorder and postorder do not uniquely define a binary tree.

Inorder And Preorder

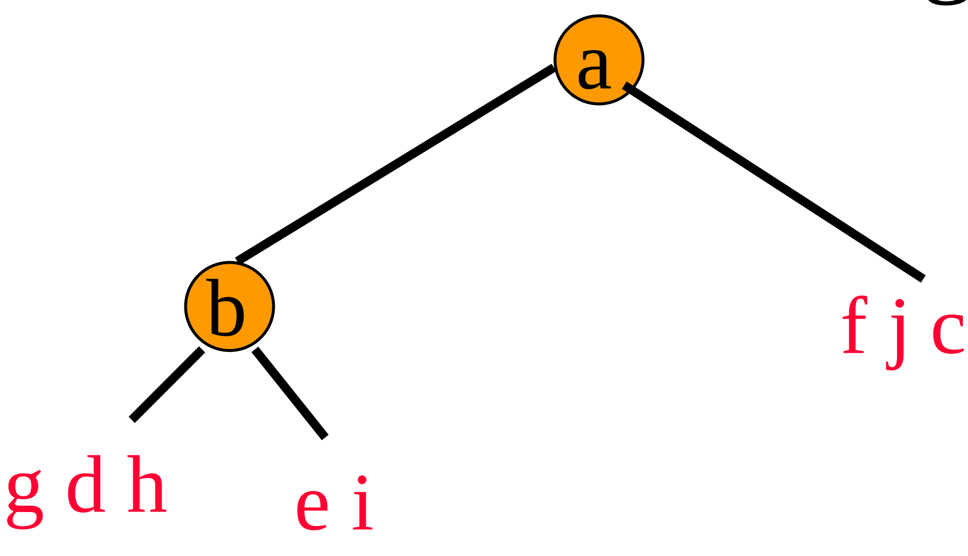
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; g d h b e i are in the left subtree; f j c are in the right subtree.



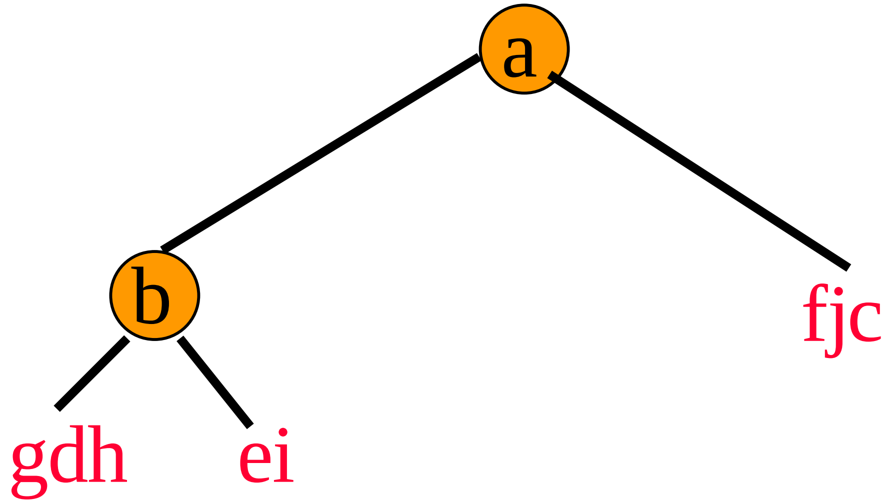
Inorder And Preorder



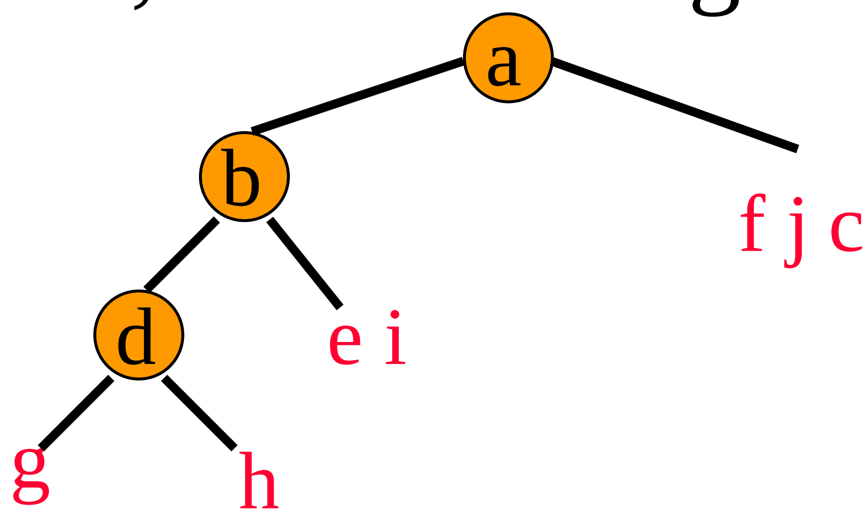
- preorder = a b d g h e i c f j
- b is the next root; g d h are in the left subtree; e i are in the right subtree.



Inorder And Preorder



- preorder = a b d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.



Example-1

Let us consider the below traversals:

Inorder sequence: D B E A F C

Preorder sequence: A B D E C F

Construct Tree from given Inorder and Preorder traversals.

Solution

```
       A                        A
      / \                      / \
     /   \                    /   \
D B E     F C                B     C
                            / \   /
                           /   \ /
                          D    E F
```

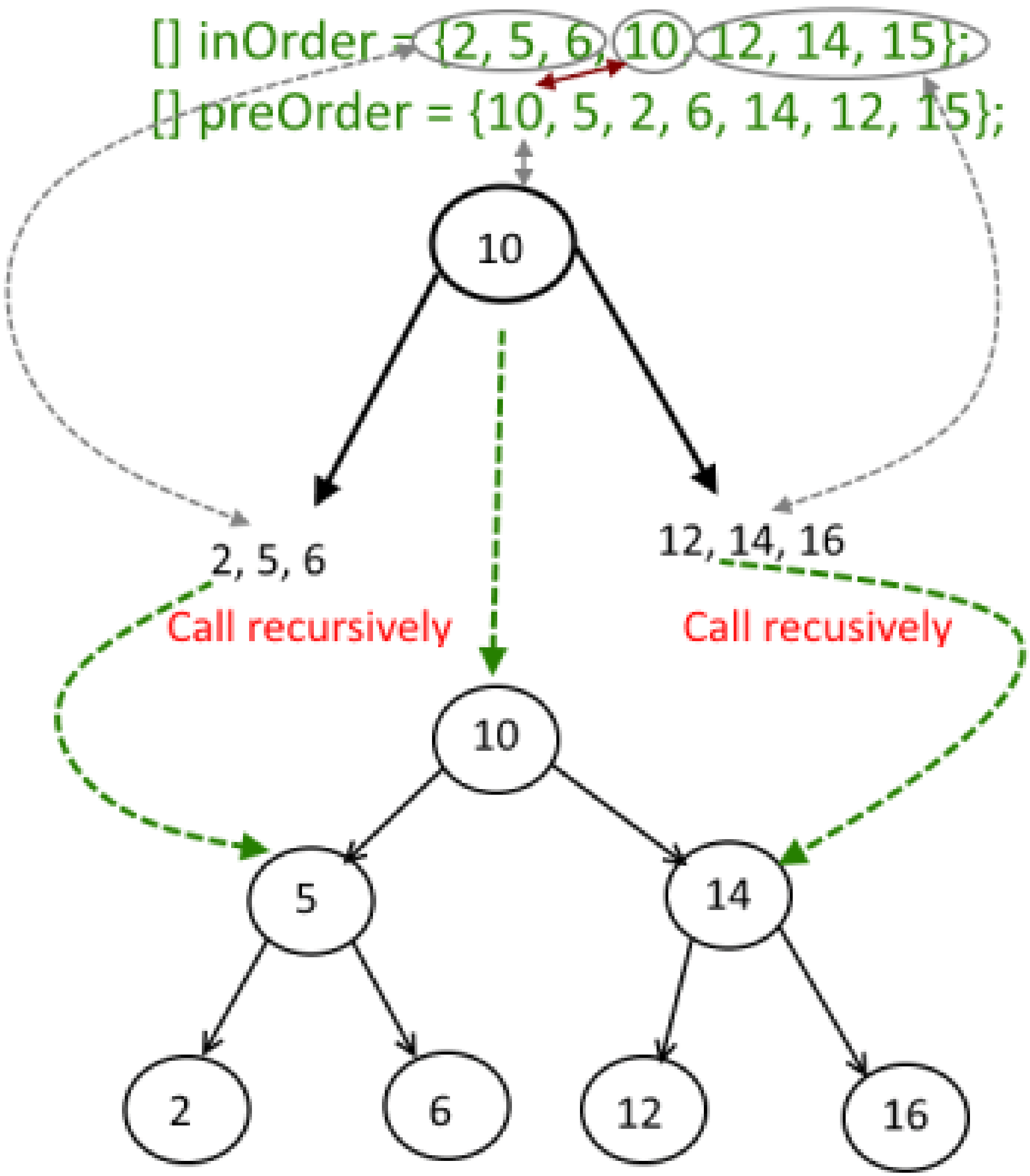
Example-2

InOrder = {2,5,6,10,12,14,15};

PreOrder = {10,5,2,6,14,12,15};

Construct Tree from given Inorder and Preorder traversals.

Solution


[] inOrder = {2, 5, 6, 10, 12, 14, 15};
[] preOrder = {10, 5, 2, 6, 14, 12, 15};
10
2, 5, 6
12, 14, 16
Call recursively
Call recusively
10
5
14
2
6
12
16

Inorder And Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; g d h b e i are in left subtree; f j c are in right subtree.

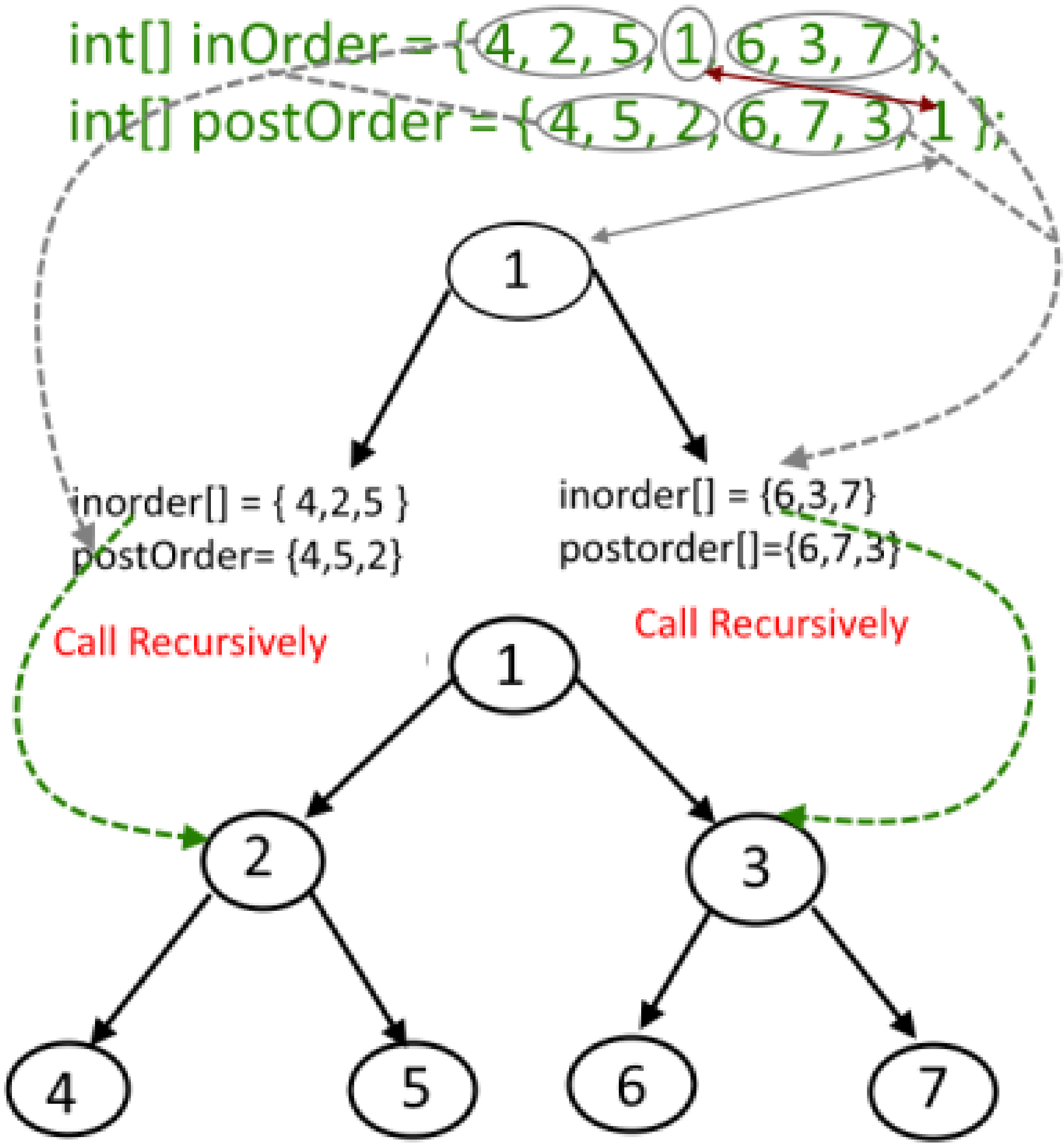
Example-1

inOrder = { 4, 2, 5, 1, 6, 3, 7 };

postOrder = { 4, 5, 2, 6, 7, 3, 1 };

Construct Tree from given Inorder and Preorder traversals.

Solution


int[] inOrder = {4, 2, 5, 1, 6, 3, 7 };
int[] postOrder = {4, 5, 2, 6, 7, 3, 1 };
1
inorder[] = { 4,2,5 }
postOrder= {4,5,2}
inorder[] = {6,3,7}
postorder[]={6,7,3}
Call Recursively
Call Recursively
1
2
3
4
5
6
7